CRYPTANALYSIS OF THE SODARK FAMILY OF CIPHER ALGORITHMS

by

Marcus Dansarie

September 2017

Thesis Advisor:	David Canright
Second Reader:	Raymond R. Buettner Jr.

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704–0188*

1. AGENCY USE ONLY *(Leave Blank)*	**2. REPORT DATE** September 2017	**3. REPORT TYPE AND DATES COVERED** Master's Thesis 09-26-2016 to 09-22-2017

4. TITLE AND SUBTITLE CRYPTANALYSIS OF THE SODARK FAMILY OF CIPHER ALGORITHMS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Marcus Dansarie	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

The thesis studies the security of the SoDark family of cipher algorithms through cryptanalysis. The ciphers in question are used to protect messages sent by second- and third-generation automatic link establishment (ALE) systems for high frequency radios. Radios utilizing ALE technology are in use by a multitude of government and non-government organizations worldwide. Structural attacks on up to eight rounds based on differential properties are presented and implemented in practice. An efficient logic circuit representation of the only nonlinear component of the ciphers, the S-box, is generated. That representation, converted to conjunctive normal form (CNF), is used to perform key-recovery attacks on up to four rounds with the use of Boolean satisfiability problem (SAT) solvers. The logic circuit representation is further used to develop an efficient bitslicing CUDA implementation of the cipher. Its efficiency in attacking the cipher is demonstrated. The impact of the attacks on the ALE system is considered. Finally, the thesis includes suggestions regarding a replacement cipher and ideas for further cryptanalysis.

14. SUBJECT TERMS cryptanalysis, automatic link establishment, ALE, high frequency radio, HF radio, block ciphers, SoDark, electronic warfare, algebraic attacks	**15. NUMBER OF PAGES** 115
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 **Standard Form 298 (Rev. 2–89)** **Prescribed by ANSI Std. 239–18**

THIS PAGE INTENTIONALLY LEFT BLANK

CRYPTANALYSIS OF THE SODARK FAMILY OF CIPHER ALGORITHMS

Marcus Dansarie
Sub-lieutenant, Swedish Armed Forces
B.S., Swedish Defence University, 2009

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION WARFARE SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2017

Approved by: David Canright
Thesis Advisor

Raymond R. Buettner Jr.
Second Reader

Dan C. Boger
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The thesis studies the security of the SoDark family of cipher algorithms through cryptanalysis. The ciphers in question are used to protect messages sent by second- and third-generation automatic link establishment (ALE) systems for high frequency radios. Radios utilizing ALE technology are in use by a multitude of government and non-government organizations worldwide. Structural attacks on up to eight rounds based on di˙erential properties are presented and implemented in practice. An eÿcient logic circuit representation of the only nonlinear component of the ciphers, the S-box, is generated. That representation, converted to conjunctive normal form (CNF), is used to perform key-recovery attacks on up to four rounds with the use of Boolean satisfiability problem (SAT) solvers. The logic circuit representation is further used to develop an eÿcient bitslicing CUDA implementation of the cipher. Its eÿciency in attacking the cipher is demonstrated. The impact of the attacks on the ALE system is considered. Finally, the thesis includes suggestions regarding a replacement cipher and ideas for further cryptanalysis.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

2DES	Double DES
3DES	Triple DES
2G	second-generation
3G	third-generation
4G	fourth-generation
AES	Advanced Encryption Standard
AL	application level
ALE	automatic link establishment
ANF	algebraic normal form
ASCII	American Standard Code for Information Interchange
ASIC	application-specific integrated circuit
BCD	binary-coded decimal
CNF	conjunctive normal form
CPU	central processing unit
CRC	cyclic redundancy check
DDT	difference distribution table
DES	Digital Encryption Standard
DIMACS	Center for Discrete Mathematics and Theoretical Computer Science
EFF	Electronic Frontier Foundation

EM	Even–Mansour
FOT	frequency of optimum transmission
FPGA	field-programmable gate array
GPGPU	general-purpose computing on graphics processing units
GPU	graphics processing unit
GSM	Global System for Mobile Communications
HF	high frequency
LAT	linear approximation table
LUF	lowest usable frequency
LUT	lookup table
MHz	megahertz
MUF	maximum usable frequency
MITM	meet-in-the-middle
MQ	multivariate quadratic
NP	non-deterministic polynomial-time
NPS	Naval Postgraduate School
NSA	National Security Agency
OpenCL	Open Computing Language
PDU	protocol data unit
PI	protection interval
PTX	Parallel Thread Execution
SAT	Boolean satisfiability problem

SIMD	single instruction, multiple data
TMTO	time–memory trade-off
UTC	Coordinated Universal Time
VOACAP	Voice of America Coverage Analysis Program
XOR	exclusive or

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: Introduction

1.1 Introduction

In radio communications, the frequencies between 3 megahertz (MHz) and 30 MHz are commonly referred to as the high frequency (HF) band. The main advantage of using this frequency band for communication is that it allows global coverage without any infrastructure. This is due to those frequencies' ability to reflect off the ionosphere. Therefore, a notable characteristic of the HF band is that propagation conditions change continuously with changes in the ionosphere. The properties of the ionosphere are heavily dependent on a number of factors, including time of day and year, geographic location, and the sun's 11-year cycle. When establishing a link, i.e., calling another radio station in order to transfer information, all these factors and more must be taken into account when selecting transmission parameters such as frequency and power [1]. To achieve reliable communications on the HF radio bands, skilled and experienced operators are therefore normally needed.

In the past few decades, advances in automatic link establishment (ALE) technology have allowed relatively unskilled operators to operate HF radios and establish communication links with success rates and times close to those of skilled and experienced operators. The addition of automation to any system inevitably introduces both new security issues as well as new variants of previous issues. The second-generation (2G) and third-generation (3G) ALE standards address this by including an option for encrypting the link establishment messages that are sent over the air [1].

1.2 Purpose and Motivation

The purpose of this thesis is to study the security of the SoDark family of ciphers that are used to encrypt link establishment messages in the 2G and 3G ALE standards. The security provided by the ciphers directly affects the performance of ALE systems in the presence of adversarial electronic warfare measures, which makes knowledge of their security bounds important.

The SoDark algorithms have been developed specifically for the ALE application [1]. No public cryptanalysis of the algorithms is available, so their security is, in effect, unknown. Both 2G and 3G ALE systems are in active use worldwide by users ranging from government and military, to non-governmental organizations and amateur radio operators [2]. If cryptographic weaknesses exist in the ciphers protecting these users' ALE HF communications, knowledge of those weaknesses might help the users compensate for those weaknesses and, eventually, eliminate them.

1.3 Methodology

The bounds of security of ciphers are established through cryptanalysis, described in Chapter 2. For academic purposes, any weakness in a cryptographic system is enough for it to be considered broken. This includes attacks that are infeasible in practice or only possible under very special circumstances. A cipher is considered broken in practice if an attack that affects the security provided by the cipher can be performed in some real-life setting [3].

As such, the method employed in academic cryptanalysis is that of hypothesis testing. The null hypothesis, then, is that the cipher is secure and that the most efficient way to attack it is through an exhaustive key search (see Chapter 2). An attack on the cipher that requires less effort than this constitutes a falsification of the null hypothesis.

1.4 Thesis Outline

Chapter 2 provides a brief theoretical background on a number of concepts in cryptography and information security that are central to the material covered in the rest of the thesis. The chapter also includes a brief overview of ALE technology.

Chapter 3 contains a description of the SoDark family of ciphers, mainly based on the specifications in [1], [4], and [5]. It also introduces the mathematical notation used in the cryptanalysis of the ciphers. The chapter also investigates the properties and selection criteria of the SoDark S-box and generalizes the cipher's structure to the Even–Mansour (EM) construction. A brief investigation of the cipher's properties with regard to linear cryptanalysis is also performed.

Chapter 4 contains the main contributions of the thesis: differential-based structural key recovery attacks on up to eight rounds of the 24-bit SoDark-3 algorithm.

Chapter 5 describes the process of generating efficient logic circuit representations of the SoDark S-box. The logic circuit representations are used in the attacks presented in Chapters 6 and 7.

Chapter 6 describes the conversion of the logic circuit representations from Chapter 5 into conjunctive normal form (CNF) and the use of Boolean satisfiability problem (SAT) solvers for key recovery attacks on up to four rounds of SoDark-3.

Chapter 7 describes the development of a high-performance bitslicing CUDA implementation for brute force key recovery attacks on the full cipher. Conversion of the developed known-plaintext attack into a ciphertext-only attack is described.

Chapter 8 concludes the thesis with a summary of the main results. It investigates the consequences of the results on the ALE system and provides recommendations. A replacement cipher, based on best practices, is suggested. The chapter finishes with a brief description of possible areas of study for further cryptanalysis of the SoDark cipher family.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background

2.1 Information Security

Information security is a term used for the practices concerning the protection of information, regardless of its physical form. The notion of protection is primarily expressed in three core concepts: confidentiality, integrity, and availability. Other concepts such as non-repudiation, accountability, reliability, or variants thereof are sometimes included as further core concepts. Here, however, focus is on the three primary concepts, which are defined as

- Confidentiality is the protection of the information content itself so that only those authorized are able to access and use it.
- Integrity is the protection of information against unauthorized change as well as the ability to detect unauthorized changes that have been made.
- Availability is the protection of the ability to access the information so that it is available for authorized users to read or modify. As such, the protection is against physical loss of the information itself as well as against loss of the ability to access or transfer it.

Methods for achieving the three aforementioned conditions vary depending on the consequences of failure to protect information as well as the information's physical form. They can include legislation, physical obstacles, backups, spare systems, training, and authentication mechanisms as well as mathematical and computer algorithms such as cryptosystems [6].

2.2 Block Ciphers

Block ciphers are prevalent as fundamental building blocks of other algorithms or protocols that aim to provide confidentiality, integrity, or availability in digital systems. In that regard, they are known as cryptographic primitives. Their basic purpose is to provide a means of transforming messages between plaintext space and ciphertext space using a secret key. To do this, a block cipher specifies an encryption function E and a decryption function $D = E^{-1}$ that take the key and message as parameters. In other words: $C = E_K(P)$ and

$P = D_K(C)$. Note, that for this to work, E_K must be bijective so that its inverse D_K exists and $D_K(E_K(P)) = P \;\; \forall K, P$. The relationship between E, D, P, C, and K is shown schematically in Figure 2.1.

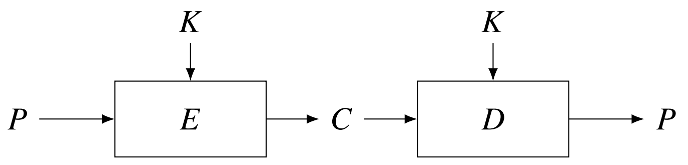

Figure 2.1. A generic block cipher with encryption function E and decryption function D.

In digital block cipher systems, the sets of plaintexts and ciphertexts consist of all binary strings of a certain length n: $P, C \in \{0,1\}^n$. This length is known as the block size. The key is also a binary string of fixed length: $K \in \{0,1\}^k$, but there is no requirement for the key size k to be the same as the block size n. Nevertheless, this is the case for some ciphers such as the Advanced Encryption Standard (AES), when it is used with a 128-bit key [7].

A block cipher provides security by making it computationally infeasible to discover the plaintexts of any number of given ciphertexts, or discover the key used to generate them. The inverse, calculating the ciphertexts of any number of given plaintexts, should also be infeasible. This is perhaps the most important requirement for any cipher—that the security must rely only on the key. In other words, knowledge of the cipher algorithm or any number of plaintexts or ciphertexts should not allow an attacker to gain any more information about the key or unknown plaintexts or unknown ciphertexts. This principle is known as Kerckhoffs' principle [8] and is a fundamental requirement in all modern cryptography.

The properties that ciphers must have in order to be secure are described in Shannon's seminal paper [9]. In particular, he introduces the two principles of diffusion and confusion that are used to prevent statistical analysis. Diffusion means that any properties of parts of the plaintext should be spread out over as much of the ciphertext as possible. In block ciphers, this means that the avalanche effect is a desirable property. That is, the change in a single bit of the plaintext should cause the probability of change for any given bit of ciphertext to be $\frac{1}{2}$ [7]. While demonstration of the avalanche effect shows a cipher has diffusion, it is not enough to prove any level of security. For this, confusion is also necessary, which is in essence the same statement for the key—no simple statistical relation between the key and ciphertext should exist. Both diffusion and confusion are necessary for a cipher to be secure. A cipher without one or the other is likely to be vulnerable to statistical attacks.

The fact that the security of a block cipher should only be dependent on the key makes the size of the key space important. If the key space is too small, an adversary that has access to a small number of ciphertexts and their corresponding plaintexts can simply perform trial decryption with all possible keys until the correct one is found. Note that, by the pigeonhole principle, if the key size is larger than the block size, then, for any plaintext, there exists at least one ciphertext that is generated by more than one key and vice versa.

As mentioned previously, block ciphers are usually not used to directly encrypt messages in block-sized chunks. Instead, they are used as cryptographic primitives in block cipher modes of operation. These modes prevent certain security issues associated with using block ciphers directly, enable encryption of variable-length messages, and provide other desirable properties such as authentication [7]. While modes of operation are very important in the larger context of the use of block ciphers, they have no bearing in the context of the usage of the ciphers studied in this thesis.

Some block ciphers have a third input to the encryption function, in addition to the key and plaintext, called a tweak. The first widely known cipher algorithm to use a tweak was probably the Hasty Pudding AES candidate [10]. A tweak provides additional keying bits that, unlike the key, are not necessarily secret [11]. The tweak is normally stored or sent along with the plaintext. The purpose of a tweak is to improve the security of the cipher with the additional non-secret bits. Ideally, no two plaintexts should be encrypted with the same combination of key and tweak. The cipher must still be secure even if that is the case, as the tweak input may not be used at all in some applications. Worse, it could be controlled by an adversary. Like the other inputs to a block cipher, the output should exhibit the avalanche property with respect to the tweak.

2.3 Automatic Link Establishment Systems

As mentioned in Section 1.1, the performance of HF radio systems is highly dependent on ionospheric conditions. The most important factors affecting the properties of the ionosphere with respect to HF radio are: time of day and year, geographic location, and the sun's 11-year cycle. Additionally, equipment parameters such as output power, antennas, and selected modulation also affect propagation. Figure 2.2 shows a HF radio propagation diagram generated with the Voice of America Coverage Analysis Program (VOACAP) [12]. The diagram shows maximum usable frequency (MUF), lowest usable frequency (LUF), and frequency of optimum transmission (FOT) for communication between two geographic locations during different times of day. It should be apparent that propagation conditions change over time.

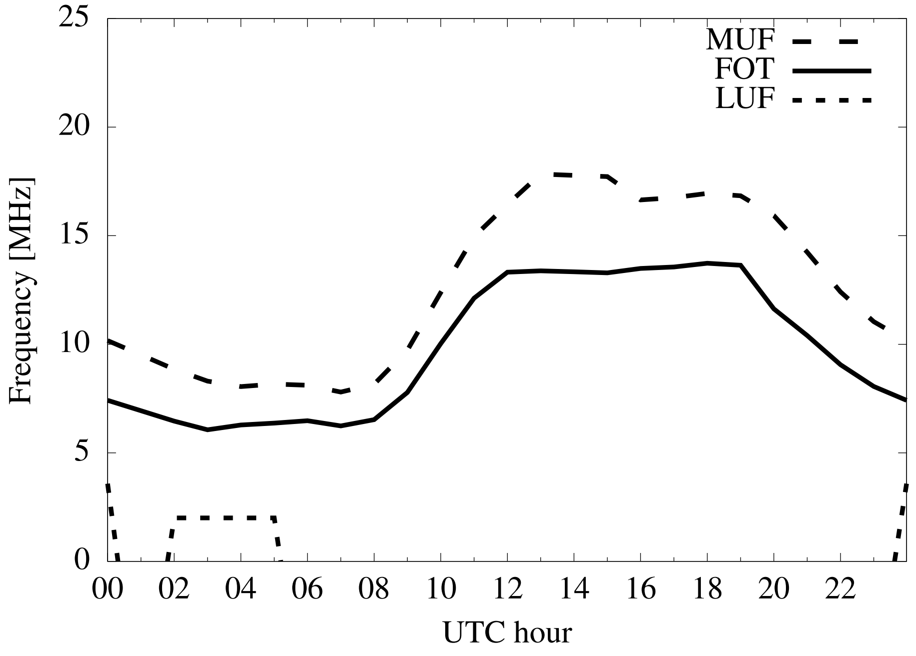

Figure 2.2. Example HF propagation diagram showing maximum usable frequency (MUF), lowest usable frequency (LUF), and frequency of optimum transmission (FOT) between Grimeton, Sweden, and Long Island, New York, during September 2017. Produced using VOACAP.

Understanding and utilizing the ionospheric conditions correctly as an HF radio operator requires training and experience. ALE technologies were created to offset most of this need with technology. In an ALE system, a computer selects transmission parameters such as frequency and power using a model of the ionosphere fed with a large number of parameters. In addition, some ALE systems perform regular soundings where one or more stations in an ALE radio network transmit sounding signals that are used by receiving stations to measure current propagation conditions on different frequencies, thereby improving the model's predictive accuracy [1].

The first ALE systems were proprietary developments by a number of commercial vendors. Interoperability suffered as a consequence. In response to this, 2G ALE was developed and standardized in MIL-STD-188-141 [4] and FS-1045 [13]. This enabled interoperability between radios from different manufacturers as well as between organizations [1].

Radios in ALE systems exchange messages in the form of protocol data units (PDU). All 2G ALE PDUs are exactly 24-bits long and consist of a three-bit preamble and three seven-bit ASCII characters. A typical call from one 2G ALE radio to another with a request to establish a communications link will consist of three PDUs. The first two are identical and contain the intended receiver's address while the third contains the sender's address. For example, the first and second would contain the preamble <`TO`> (010 in binary) followed by a three ASCII character address, such as `SAM`. This example would be hex encoded as `54e0cd`. The third PDU in this example could contain the preamble <`TIS`> (101 in binary) followed by the sender address `JOE` and be hex encoded as `b2a7c5` [5].

An obvious requirement for two radios to be able to communicate is that the sender transmits on the same frequency as the one on which the receiver is listening. To adopt to varying transmission conditions, ALE radio networks must use several different frequencies. This is achieved by having all idle radios in a network scan a predefined list of frequencies by sequentially tuning to them for a short period of time, called the dwell time, and listening for ALE PDUs. A radio that needs to establish a link with another radio selects a suitable frequency using the ionospheric transmission model and starts transmitting PDUs on that frequency. Because the radios scan frequencies asynchronously, some number of tries will be needed for the intended receiver to register the transmission. When a radio detects a PDU intended for it, it stops scanning and transmits a reply.

The asynchronous scanning is a source of some inefficiency. The next generation of the standard, 3G ALE, solved this problem utilizing synchronous scanning, i.e., all radio stations in the network tune to the same frequency at the same time. Since a transmitting radio knows which frequency an intended receiver is tuned to at any instant, only a single transmission will normally be required. A requirement for this to work is for all stations' internal clocks to be synchronized with an accuracy less than the dwell time. This can be done by manual input, with the help of external timing input from a global navigation satellite system receiver, or through asynchronous over-the-air synchronization with another ALE station.

3G ALE uses 26- and 48-bit PDUs that have different formats from 2G ALE. The addressing format is different as well, with 3G ALE using binary addresses. Additionally, 3G ALE PDUs contain cyclic redundancy check (CRC) checksums, allowing for error detection.

To prevent unauthorized users from linking with radios in an ALE radio network, or to recover information from intercepted PDUs, the standards specify an optional linking protection scheme that allows for encryption of transmitted PDUs. ALE linking protection has five application levels (AL): AL-0 through AL-4. Their definitions from [4] are shown in Table 2.1.

Table 2.1. ALE linking protection application levels. Adapted from [4].

Application level	Definition
AL-0	unprotected application level
AL-1	unclassified application level
AL-2	unclassified enhanced application level
AL-3	unclassified but sensitive application level
AL-4	classified application level

The first application level, AL-0, corresponds to all encryption being turned off. Application levels AL-1 and AL-2 use the SoDark cipher algorithms and are described as "for general U.S. Government and commercial use." The difference between AL-1 and AL-2 is that the latter uses a shorter protection interval (PI): two seconds instead of 60 seconds. The tweak (see Section 2.2 and Chapter 3) used for encryption of PDUs remains the same for the duration of a PI. This makes AL-1 somewhat vulnerable to replay attacks.

AL-3 and AL-4 use hardware cipher modules developed and approved by the National Security Agency (NSA). AL-4 is the only AL intended for the protection of classified information. These application levels are outside the scope of this thesis.

The tweak, which is referred to as *seed* in the standards, is a 64-bit value used to prevent replay attacks. Chapter 3 describes how the tweak is used by the linking protection cipher in the ALE protocols. It contains the transmission frequency, PI number (i.e., transmission time), date, and the word number (i.e., the order of the PDU in the current transmission). The advantage of using that data is that it is implicitly known by the receiver and does not need to be transferred along with the ciphertext. Table 2.2 shows the tweak data structure.

Table 2.2. Construction of tweak used in ALE linking protection. Bit number 1 is the most significant bit and 64 the least significant. Adapted from [4].

Field	Month	Day	PI	Word number	Zero pad	Frequency (BCD)
Bits	1 – 4	5 – 9	10 – 26	27 – 34	35 – 36	37 – 64

2.4 Cryptanalysis

Cryptanalysis is, as the name implies, the analysis of cryptosystems. In particular, cryptanalysis normally aims to establish the bounds of a cryptosystem's security. It is practiced by both users of cryptosystems and their adversaries. Cryptosystem users perform cryptanalysis to ensure there are no ways to recover information about plaintexts, ciphertexts, or keys. Their adversaries do cryptanalysis in the hope of finding such ways [7].

In general, any ability to recover information that requires less effort than trying, on average, half of the possible keys is considered a break for cryptanalytic purposes. For example, there exists a key recovery attack for AES with computational complexity proportional to $2^{126.1}$, while the average computational complexity of a brute force attack is 2^{127}. AES is therefore broken in theory. Since the complexity of this attack is still astronomical, however, and requires a very large amount of data, the cipher is not broken in practice and is still considered safe to use [14].

The starting point for cryptanalysis on a particular cipher is usually to study its mathematical description and to apply cryptanalytic techniques that have been successful with other similar ciphers. A common approach is to start by analyzing versions of the cipher with reduced security. This can, for example, be a version that has a reduced number of rounds or in an improbable setting, such as having the entire codebook for a given key. The insights from these attacks may then provide tools and methods for attacking the cipher with more rounds or in more generalized settings [3].

Perhaps the single most important property for a cipher to have if it is to be resistant to cryptanalysis is nonlinearity. This property follows directly from Shannon's diffusion and confusion properties. A completely linear cipher can simply be described as a system of linear equations that can be solved by Gaussian elimination given a very small number of plaintexts. Since systems of linear equations can be solved in polynomial time, this is expected to be faster than an exhaustive search of the key space, even for very small key spaces.

A nonlinear cipher on the other hand, must be described as a system of equations of higher order. Such a system of equations is reducible to the multivariate quadratic (MQ) problem, which is non-deterministic polynomial-time (NP)-hard. Thus, it has complexity $O(2^{\alpha n})$, $0 < \alpha \leq 1$ in the case of n binary variables. In most cases, this makes it harder to attack a cipher this way than the brute force approach of testing all the keys, which has complexity equivalent to 2^{k-1} encryptions on average, where k is the key length in bits.

There are exceptions: In some cases, it is possible to linearize the system of equations, i.e., to replace all nonlinear terms with new variables and then solve the resulting system of linear equations. This will yield a number of spurious solutions that must be filtered out. Linearization of a MQ system of equations is only possible if it is sufficiently sparse and overdefined. Another exception is the use of SAT solvers or constraint solvers to solve the system of equations. SAT solvers are able to solve large systems of Boolean equations with comparatively high speed [15].

While many attacks are specific to certain ciphers, there are a number of attacks that work on large classes of block ciphers. Some examples of such generic attacks are given in the following sections.

2.4.1 Brute Force Attacks

A brute force attack works by trying every possible key until the right one is found. On average, half the key space needs to be searched before the correct key is found so, for a k-bit key, the effort is proportional to 2^{k-1}. The only protection against brute force key search is to ensure that the key space is large enough for the attack to be intractable, at least during the expected period for which the encrypted data needs to be protected. In general, a cipher is considered secure if no attacks exist that are faster than an exhaustive search in practice and the size of the key space makes a brute force search impossible. Various recommendations for minimum key lengths exist. Table 2.3 compiles the recommendations from [16] and [17]. Among the sources consulted, there is consensus that a 128-bit key size provides good security, 64 bits or less provides no security in practice, and 80 bits is the smallest key size that provides any measure of security.

Table 2.3. Key length recommendations. Adapted from [16], [17].

	Level of security	
Key size (bits)	Knudsen & Robshaw (2010)	ECRYPT II (2012)
32		attacks in real-time by individuals
40	easy to break	very short-term protection
64	practical to break	
80	not currently feasible	smallest general-purpose level
96		legacy standard level
112		medium-term protection
128	very strong	long-term protection
256	exceptionally strong	foreseeable future

An efficient brute force attack requires an efficient implementation of the cipher function. Application-specific integrated circuits (ASIC) built specifically for breaking the cipher in question is the fastest, but most expensive, technology. Constructing an ASIC to perform brute force key search requires custom integrated circuit design and manufacturing, which is expensive and out of reach for individuals and small organizations. In 1998, the Electronic Frontier Foundation (EFF) built an ASIC-based computer, *Deep Crack*, that could break the 56-bit Digital Encryption Standard (DES) cipher in less than a week. The budget for the project was about 200,000 U.S. dollars [18]. This is an example of a medium-size organization's ability to break 56-bit ciphers in the late 1990s.

A slower and cheaper, but still quite efficient, way to perform a brute force search is to employ field-programmable gate arrays (FPGA). FPGAs are reconfigurable hardware gate networks that enable efficient implementations and parallelization of calculations at comparatively low cost. Cloud FPGA computing services as well as FPGA expansion cards for personal computers are available. This could enable the use of FPGAs for brute force key searches by individuals and organizations of any size.

Graphics processing units (GPU) are primarily designed for real-time rendering of graphics on personal computers. Yet, their design also makes them useful for highly parallel computation—a single modern GPU can contain thousands of processor cores. This has led to the emergence of general-purpose computing on graphics processing units (GPGPU) programming frameworks, such as OpenCL and CUDA, specifically tailored for GPU computing. These frameworks are used to write programs that solve various hard problems encountered in a wide range of fields.

Lastly, brute force key search can be done with central processing units (CPU) in general purpose computers. Except for ciphers that have been specifically engineered to resist the aforementioned methods, this tends to be the slowest method. To their advantage, however, are shorter development time and the possibility of using existing software implementations of the cipher. In addition, an organization can use the computer infrastructure it already has in place to perform the key search. There are also examples of the Internet being used to leverage the power of computers all over the world to perform brute force key search.

Regardless of the hardware used, the fastest implementations of ciphers are in forms that regard the cipher as a network of logic gates rather than as an imperative computer program. In ASICs and FPGAs, this enables a design that, in effect, tests one key per clock cycle. In GPUs and CPUs, this enables bitslicing implementations. In a bitslicing implementation, each variable in the program represents one bit of state and the entire cipher is implemented in software as bitwise logic operations. This enables instruction level parallelism, where every instruction operates on a number of parallel encryptions or decryptions. The exact number is dependent on the platform's register size. With modern processors that have single instruction, multiple data (SIMD) instruction sets with registers as wide as 256 or 512 bits, this means that that many encryptions or decryptions can be performed in parallel on a single processor core. Additionally, bit level permutations are performed at no cost at all in bitslicing implementations [19].

Finding the most efficient logic gate representation of nonlinear parts of the cipher, such as S-boxes, is an NP-hard problem. Without a clear mathematical description of an S-box, a partial search of the solution space using a heuristic algorithm may be the only way to find an efficient, but non-optimal, solution.

2.4.2 Time–Memory Trade-Off Attacks

Time–memory trade-off (TMTO) attacks exist for all block ciphers. The simplest example is to construct a dictionary that associates any given plaintext–ciphertext pair with a key. For most ciphers that are in practical use, the storage space required to mount such an attack makes this impossible. For a cipher with a block and key size of n bits, the storage space required for the lookup table would be $n \cdot 2^{2n}$ bits. The required space can be reduced to $n \cdot 2^n$ bits if the dictionary is restricted to a single plaintext.

Hellman [20] describes a TMTO attack that allows the attacker to choose an almost arbitrary point on a trade-off curve between the extremes provided by the brute force and dictionary attacks. A TMTO attack starts by creating reusable tables for a certain plaintext by performing precomputations of a complexity equivalent to the brute force recovery of a single key. Given a ciphertext corresponding to that plaintext, the key can be recovered by quickly regenerating only parts of the precomputations with the help of the tables. This way, the key can be recovered significantly faster than by brute force alone. TMTO attacks have been used to perform practical breaks of ciphers that are in current use. One of the more notable examples of a cipher broken by this is the A5/1 cipher used in the GSM standard for mobile telephony [21]. For details on the attack, the reader is referred to Hellman's original paper [20] or to the description in [16].

2.4.3 Meet-in-the-Middle Attacks

Meet-in-the-middle (MITM) attacks are an example of a type of structural attack. They exploit the fact that some ciphers can be divided into two parts, where neither part is dependent on the full key. This attack type was first described in [22], where possible improvements to the DES algorithm are investigated. When considering double encryption with DES using two different keys, the authors show that such a system can be broken with effort proportional to 2^{57}, despite the system having a 112-bit key. Thus, the double encryption with two independent keys adds only a single bit of security. As an example, Algorithm 2.1 performs a MITM attack on a product cipher $f = h \circ g$ using two known plaintext–ciphertext pairs.

Algorithm 2.1 Perform a meet-in-the-middle attack on a product cipher $f = h \circ g$. Adapted from [22].

```
1:  procedure MeetInTheMiddle($P_1, C_1, P_2, C_2$)
2:      $L \leftarrow$ empty list
3:      for all $k_1$ do
4:          $v \leftarrow g_{k_1}(P_1)$
5:          $L$.append($v, k_1$)
6:      end for
7:      for all $k_2$ do
8:          $w \leftarrow h_{k_2}^{-1}(C_1)$
9:          $k_1 \leftarrow L[w]$
10:         if $h_{k_2}\big(g_{k_1}(P_2)\big) = C_2$ then
11:             Print($k_1, k_2$)
12:         end if
13:     end for
14: end procedure
```

In the case of Double DES (2DES), we expect to find the key in about 2^{57} DES operations using 2^{56} 56-bit blocks of memory. For that reason, DES was eventually strengthened through Triple DES (3DES), which is still vulnerable to the same attack, but with an attack complexity of about 2^{112} DES operations. This was considered sufficiently prohibitive at the time.

2.4.4 Differential Cryptanalysis

Differential cryptanalysis is, together with linear cryptanalysis, one of the strongest known general attacks on block ciphers. It was first described in the open literature by Biham and Shamir [23]. Attacks based on differential cryptanalysis work with differences, called differentials, between inputs and outputs of parts of a cipher. Commonly, the differentials are defined as the bitwise XOR of two values, although other definitions such as modular addition can be used.

The basic idea of differential attacks is to distinguish the output of a certain function from random by considering the probability that a certain output differential is generated by a certain input differential or vice versa. Since S-boxes are the only source of nonlinearity in many ciphers, the study of their differential properties is usually an important part of cryptanalysis. An example 4×4 S-box from [16] is shown in Table 2.4 and Table 2.5 shows the parts of its difference distribution table (DDT) that correspond to inputs that are bitwise complements. It is clear that the differentials shown in the rightmost column are not evenly distributed. The value d, for example, appears with probability $\frac{10}{16}$ and 12 of the 16 possible values have probability 0.

Table 2.4. An example 4×4 S-box vulnerable to differential cryptanalysis. Adapted from [16].

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	6	4	c	5	0	7	2	e	1	f	3	d	8	a	9	b

Table 2.5. Excerpt from the difference distribution table of the example S-box from Table 2.4. Adapted from [16].

i	j	$S(i)$	$S(j)$	$S(i) \oplus S(j)$
0	f	6	b	d
1	e	4	9	d
2	d	c	a	6
3	c	5	8	d
4	b	0	d	d
5	a	7	3	4
6	9	2	f	d
7	8	e	1	f
8	7	1	e	f
9	6	f	2	d
a	5	3	7	4
b	4	d	0	d
c	3	8	5	d
d	2	a	c	6
e	1	9	4	d
f	0	b	6	d

The DDTs of the S-boxes in a cipher, together with knowledge of its round structure, can be used to construct relations between inputs and outputs of a number of consecutive rounds that have probabilities much higher or lower than expected for a cipher adhering to Shannon's diffusion property.

2.4.5 Linear Cryptanalysis

Linear cryptanalysis was first described by Matsui in his cryptanalysis of the DES cipher [24]. As with differential cryptanalysis, it provides a method for discovering and using non-random statistical properties of the cipher. This time, the property used is the parity of certain bit positions in the input and output.

Again, an example 4×4 S-box from [16], shown in Table 2.6, illustrates the concept. The top two rows in the table show the S-box, while the additional two bottom rows show the parity of certain bits of its input and output, respectively, selected by the masks $\alpha = (1, 0, 0, 1)$ and $\beta = (0, 0, 1, 0)$. The two bottom rows differ in all columns, except for $x = $ `1` and $x = $ `f`. This means that the relation $(\alpha \cdot x) \oplus 1 = \beta \cdot S(x)$ holds with probability $\frac{14}{16}$, which is a significant difference from the $\frac{8}{16} = \frac{1}{2}$ probability expected from a S-box with good nonlinearity.

Table 2.6. An example 4×4 S-box vulnerable to linear cryptanalysis. Adapted from [16].

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	f	e	b	c	6	d	7	8	0	3	9	a	4	2	1	5
$\alpha \cdot x$	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
$\beta \cdot S(x)$	1	1	1	0	1	0	1	0	0	1	0	1	0	1	0	0

The equivalent to a DDT in linear cryptanalysis is the linear approximation table (LAT), which shows the deviation from the expected probability of $\frac{1}{2}$ for all pairs of input and output masks. As with differential cryptanalysis, the LAT in linear cryptanalysis is used to create relations between the parity of inputs and outputs of several consecutive cipher rounds. The relations can then be used in attacking the cipher.

For a more in-depth description of the methods of linear and differential cryptanalysis, the reader is referred to the excellent tutorials in [16] and [25].

CHAPTER 3: The SoDark Family of Algorithms

3.1 Background

The Lattice cipher algorithm is specified in [4]. It is a 24-bit block cipher that uses a 56-bit key and a 64-bit tweak. It has eight rounds and is used to encrypt 24-bit PDUs sent by the 2G ALE protocol. A version called SoDark-3 is used in the 3G ALE standard to encrypt 24-bits of the 26-bit PDUs. It is identical to the original Lattice algorithm, except that it uses 16 rounds. Since 3G ALE also uses 48-bit PDUs, SoDark-3 has been extended into a version with 48-bit block size called SoDark-6.

The cipher was developed specifically for the ALE application. The main purpose of the algorithm, according to [5], is to prevent unauthorized linking with radios that are part of an ALE radio network. The reference specifically mentions both replay attacks, where a previously sent legitimate PDU is replayed by an adversary, as well as attacks where the adversary is actively trying to recover the key.

Further insight is given by [1], which lists the following seven design requirements for the original Lattice algorithm:

(a) transparency to ALE protocols;
(b) self-synchronization;
(c) minimum impact on scanning dwell time;
(d) 24-bit block operation;
(e) channel- and time-varying;
(f) moderate computational requirements; and
(g) unclassified algorithm.

Requirements a, b, c, and d all have the same root cause in that the 2G ALE standard uses 24-bit PDUs and non-synchronous frequency scanning. A station in an HF radio network that uses ALE must be able to switch to a frequency and immediately start receiving PDUs. Since the dwell time, i.e., the time the station listens to any given frequency, is quite short, any received non-authentic PDU must not cause an interruption in scanning. The linking protection cipher is also an optional feature in the standard and must be a drop-in replacement in the sense that no more data than the 24 or 48 bits allocated in the transmission format can be used when linking protection is enabled.

Requirement e is needed if the cipher is to be semantically secure. Without this, it would be trivially vulnerable to traffic analysis and replay attacks. In particular, the short block size would enable an attacker to quickly compile relevant parts of the codebook for a given key.

The last two requirements, f and g, stem from the fact that the ALE algorithm and cipher are meant to be used by field radios.

The round function consists of S-box lookups and XOR operations, which makes the S-box the only nonlinear component of the cipher. Table 3.1 shows the S-box lookup table. Neither [1] nor [4] nor [5] describes how the S-box was generated or the criteria for its selection.

Table 3.1. The LATTICE and SODARK S-box. Adapted from [4].

	★0	★1	★2	★3	★4	★5	★6	★7	★8	★9	★A	★B	★C	★D	★E	★F
0★	9C	F2	14	C1	8E	CB	B2	65	97	7A	60	17	92	F9	78	41
1★	07	4C	67	6D	66	4A	30	7D	53	9D	B5	BC	C3	CA	F1	04
2★	03	EC	D0	38	B0	ED	AD	C4	DD	56	42	BD	A0	DE	1B	81
3★	55	44	5A	E4	50	DC	43	63	09	5C	74	CF	0E	AB	1D	3D
4★	6B	02	5D	28	E7	C6	EE	B4	D9	7C	19	3E	5E	6C	D6	6E
5★	2A	13	A5	08	B9	2D	BB	A2	D4	96	39	E0	BA	D7	82	33
6★	0D	5F	26	16	FE	22	AF	00	11	C8	9E	88	8B	A1	7B	87
7★	27	E6	C7	94	D1	5B	9B	F0	9F	DB	E1	8D	D2	1F	6A	90
8★	F4	18	91	59	01	B1	FC	34	3C	37	47	29	E2	64	69	24
9★	0A	2F	73	71	A9	84	8C	A8	A3	3B	E3	E9	58	80	A7	D3
A★	B7	C2	1C	95	1E	4D	4F	4E	FB	76	FD	99	C5	C9	E8	2E
B★	8A	DF	F5	49	F3	6F	8F	E5	EB	F6	25	D5	31	C0	57	72
C★	AA	46	68	0B	93	89	83	70	EF	A4	85	F8	0F	B3	AC	10
D★	62	CC	61	40	F7	FA	52	7F	FF	32	45	20	79	CE	EA	BE
E★	CD	15	21	23	D8	B6	0C	3F	54	1A	BF	98	48	3A	75	77
F★	2B	AE	36	DA	7E	86	35	51	05	12	B8	A6	9A	2C	06	4B

3.2 Notation

This section introduces the notation used in the descriptions and cryptanalysis of the SoDark family of ciphers in this and following chapters.

The bitwise exclusive or (XOR) operation is denoted by $\oplus$.

S-box lookups are denoted by s and inverse S-box lookups by s^{-1}.

Concatenation of variables is denoted by $\|$.

The full plaintext is denoted by $\mathcal{P}$, the full ciphertext by $\mathcal{C}$, the full key by $\mathcal{K}$, and the full tweak by $\mathcal{T}$.

The ciphers described are byte oriented. Input and output bytes to and from each round are denoted by the letters A through F, with the letter A representing the most significant byte of the state and the other letters representing the following bytes in falling order. To differentiate between state in different rounds, the superscript in parenthesis is used where $A^{(r-1)}$ represents the input to and $A^{(r)}$ the output from the rth round.

In the cryptanalysis, the state of several parallel encryptions are studied and subscripts are used to differentiate the parallel variables. For example, $A_1^{(0)}$ and $A_2^{(0)}$ represent the most significant input byte in two parallel encryptions.

Differentials, i.e., XOR differences between the same state variable in two parallel encryptions are denoted with the Δ character. Continuing the previous example, $\Delta A^{(0)}$ would be the differential of the most significant plaintext byte.

In some cases it will be convenient to study partial decryptions of the parts of a round that are not key-dependent. The notation $\hat{A}_1^{(1)}$ is used for such partial decryptions.

Use of the key and tweak is also byte oriented. A certain byte is denoted by k_1 for the key and t_1 for the tweak, starting with the number one for the most significant byte. Multiple-byte round keys are denoted by K. Different versions of tweak bytes in parallel encryptions are denoted by a comma in the subscript. For example, $t_{1,2}$ denotes the most significant tweak byte in the second parallel encryption.

3.3 24-bit Version (SoDark-3)

Each round operates on the incoming 24-bit word by splitting it into three bytes $A^{(r-1)}$, $B^{(r-1)}$, and $C^{(r-1)}$, with $A^{(r-1)}$ containing the most significant bits and $C^{(r-1)}$ the least significant. It then calculates three output bytes $A^{(r)}$, $B^{(r)}$, and $C^{(r)}$ in the following manner:

$$A^{(r)} = s\left(A^{(r-1)} \oplus B^{(r-1)} \oplus k_1\right) \tag{3.1}$$

$$C^{(r)} = s\left(C^{(r-1)} \oplus B^{(r-1)} \oplus k_2\right) \tag{3.2}$$

$$B^{(r)} = s\left(B^{(r-1)} \oplus A^{(r)} \oplus C^{(r)} \oplus k_3\right) \tag{3.3}$$

where s denotes the S-box lookup function and k_1, k_2, and k_3 are the most, middle, and least significant parts of the round key. Figure 3.1 shows the encryption process. Decryption is performed by inverting the operations:

$$B^{(r-1)} = s^{-1}\left(B^{(r)}\right) \oplus A^{(r)} \oplus C^{(r)} \oplus k_3 \tag{3.4}$$

$$A^{(r-1)} = s^{-1}\left(A^{(r)}\right) \oplus B^{(r-1)} \oplus k_1 \tag{3.5}$$

$$C^{(r-1)} = s^{-1}\left(C^{(r)}\right) \oplus B^{(r-1)} \oplus k_2. \tag{3.6}$$

The key schedule is completely linear. For each round, three bytes of key and three bytes of tweak are XORed to create a 24-bit round key. The bytes are used in order and the different lengths of the key and tweak ensure that the round keys are different.

The round keys for the first 16 rounds are listed in Table 3.2. As is apparent from the table, and assuming the tweak is known, knowledge of any round key will reveal parts of at least half of the round keys.

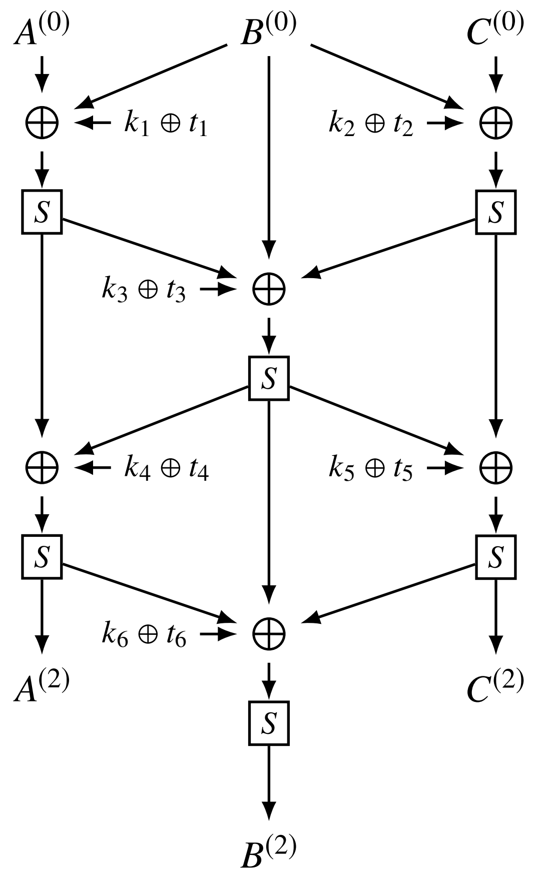

Figure 3.1. The first two rounds of the SoDark-3 algorithm.

Table 3.2. LATTICE and SODARK-3 key schedule.

Round	K_r		
	k_1	k_2	k_3
1	$k_1 \oplus t_1$	$k_2 \oplus t_2$	$k_3 \oplus t_3$
2	$k_4 \oplus t_4$	$k_5 \oplus t_5$	$k_6 \oplus t_6$
3	$k_7 \oplus t_7$	$k_1 \oplus t_8$	$k_2 \oplus t_1$
4	$k_3 \oplus t_2$	$k_4 \oplus t_3$	$k_5 \oplus t_4$
5	$k_6 \oplus t_5$	$k_7 \oplus t_6$	$k_1 \oplus t_7$
6	$k_2 \oplus t_8$	$k_3 \oplus t_1$	$k_4 \oplus t_2$
7	$k_5 \oplus t_3$	$k_6 \oplus t_4$	$k_7 \oplus t_5$
8	$k_1 \oplus t_6$	$k_2 \oplus t_7$	$k_3 \oplus t_8$
9	$k_4 \oplus t_1$	$k_5 \oplus t_2$	$k_6 \oplus t_3$
10	$k_7 \oplus t_4$	$k_1 \oplus t_5$	$k_2 \oplus t_6$
11	$k_3 \oplus t_7$	$k_4 \oplus t_8$	$k_5 \oplus t_1$
12	$k_6 \oplus t_2$	$k_7 \oplus t_3$	$k_1 \oplus t_4$
13	$k_2 \oplus t_5$	$k_3 \oplus t_6$	$k_4 \oplus t_7$
14	$k_5 \oplus t_8$	$k_6 \oplus t_1$	$k_7 \oplus t_2$
15	$k_1 \oplus t_3$	$k_2 \oplus t_4$	$k_3 \oplus t_5$
16	$k_4 \oplus t_6$	$k_5 \oplus t_7$	$k_6 \oplus t_8$

3.4 48-bit Version (SoDark-6)

The version of the algorithm with 48-bit block length, SoDark-6, is a direct extension of SoDark-3. Figure 3.2 shows the encryption process: Each round splits the incoming 48-bit word into six bytes $A^{(r-1)}$, $B^{(r-1)}$, $C^{(r-1)}$, $D^{(r-1)}$, $E^{(r-1)}$, and $F^{(r-1)}$ with $A^{(r-1)}$ containing the most significant bits and $F^{(r-1)}$ the least significant. It then calculates six output bytes in the following manner:

$$A^{(r)} = s\left(A^{(r-1)} \oplus B^{(r-1)} \oplus F^{(r-1)} \oplus k_1\right) \tag{3.7}$$

$$C^{(r)} = s\left(B^{(r-1)} \oplus C^{(r-1)} \oplus D^{(r-1)} \oplus k_2\right) \tag{3.8}$$

$$E^{(r)} = s\left(D^{(r-1)} \oplus E^{(r-1)} \oplus F^{(r-1)} \oplus k_3\right) \tag{3.9}$$

$$B^{(r)} = s\left(A^{(r)} \oplus B^{(r-1)} \oplus C^{(r)} \oplus k_4\right) \tag{3.10}$$

$$D^{(r)} = s\left(C^{(r)} \oplus D^{(r-1)} \oplus E^{(r)} \oplus k_5\right) \tag{3.11}$$

$$F^{(r)} = s\left(E^{(r)} \oplus F^{(r-1)} \oplus A^{(r)} \oplus k_6\right). \tag{3.12}$$

Again, k_i denotes the ith byte of the round key where k_1 is the most significant. The key schedule is analogous to the one used by the 24-bit versions and is shown in Table 3.3. Decryption is also analogous to the 24-bit version:

$$B^{(r-1)} = s^{-1}\left(B^{(r)}\right) \oplus A^{(r)} \oplus C^{(r)} \oplus k_4 \tag{3.13}$$

$$D^{(r-1)} = s^{-1}\left(D^{(r)}\right) \oplus C^{(r)} \oplus E^{(r)} \oplus k_5 \tag{3.14}$$

$$F^{(r-1)} = s^{-1}\left(F^{(r)}\right) \oplus E^{(r)} \oplus A^{(r)} \oplus k_6 \tag{3.15}$$

$$A^{(r-1)} = s^{-1}\left(A^{(r)}\right) \oplus B^{(r-1)} \oplus F^{(r-1)} \oplus k_1 \tag{3.16}$$

$$C^{(r-1)} = s^{-1}\left(C^{(r)}\right) \oplus B^{(r-1)} \oplus D^{(r-1)} \oplus k_2 \tag{3.17}$$

$$E^{(r-1)} = s^{-1}\left(E^{(r)}\right) \oplus D^{(r-1)} \oplus F^{(r-1)} \oplus k_3. \tag{3.18}$$

One notable change from SoDark-3 is that the mixing of inputs "wraps around" in the sense that the most and least significant bytes A and F are mixed with each other.

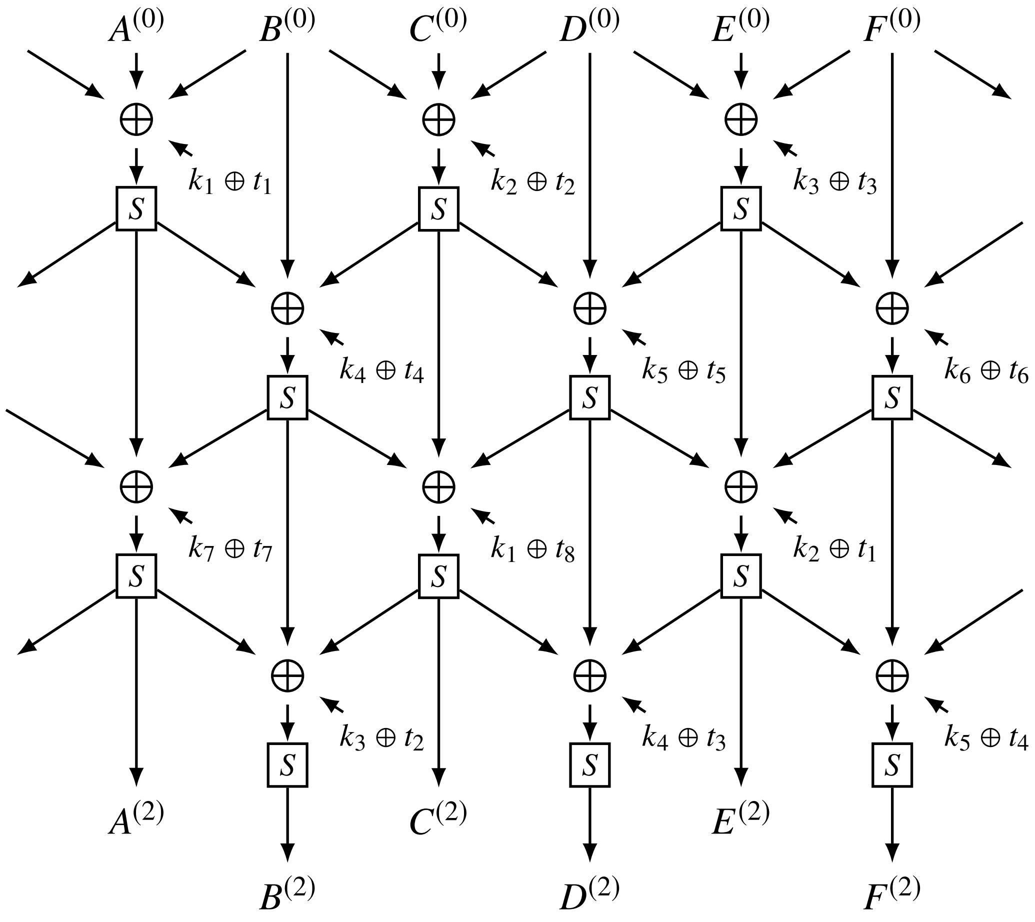

Figure 3.2. The first two rounds of the SoDark-6 algorithm.

Table 3.3. SoDark-6 key schedule.

Round	K_r					
	k_1	k_2	k_3	k_4	k_5	k_6
1	$k_1 \oplus t_1$	$k_2 \oplus t_2$	$k_3 \oplus t_3$	$k_4 \oplus t_4$	$k_5 \oplus t_5$	$k_6 \oplus t_6$
2	$k_7 \oplus t_7$	$k_1 \oplus t_8$	$k_2 \oplus t_1$	$k_3 \oplus t_2$	$k_4 \oplus t_3$	$k_5 \oplus t_4$
3	$k_6 \oplus t_5$	$k_7 \oplus t_6$	$k_1 \oplus t_7$	$k_2 \oplus t_8$	$k_3 \oplus t_1$	$k_4 \oplus t_2$
4	$k_5 \oplus t_3$	$k_6 \oplus t_4$	$k_7 \oplus t_5$	$k_1 \oplus t_6$	$k_2 \oplus t_7$	$k_3 \oplus t_8$
5	$k_4 \oplus t_1$	$k_5 \oplus t_2$	$k_6 \oplus t_3$	$k_7 \oplus t_4$	$k_1 \oplus t_5$	$k_2 \oplus t_6$
6	$k_3 \oplus t_7$	$k_4 \oplus t_8$	$k_5 \oplus t_1$	$k_6 \oplus t_2$	$k_7 \oplus t_3$	$k_1 \oplus t_4$
7	$k_2 \oplus t_5$	$k_3 \oplus t_6$	$k_4 \oplus t_7$	$k_5 \oplus t_8$	$k_6 \oplus t_1$	$k_7 \oplus t_2$
8	$k_1 \oplus t_3$	$k_2 \oplus t_4$	$k_3 \oplus t_5$	$k_4 \oplus t_6$	$k_5 \oplus t_7$	$k_6 \oplus t_8$
9	$k_7 \oplus t_1$	$k_1 \oplus t_2$	$k_2 \oplus t_3$	$k_3 \oplus t_4$	$k_4 \oplus t_5$	$k_5 \oplus t_6$
10	$k_6 \oplus t_7$	$k_7 \oplus t_8$	$k_1 \oplus t_1$	$k_2 \oplus t_2$	$k_3 \oplus t_3$	$k_4 \oplus t_4$
11	$k_5 \oplus t_5$	$k_6 \oplus t_6$	$k_7 \oplus t_7$	$k_1 \oplus t_8$	$k_2 \oplus t_1$	$k_3 \oplus t_2$
12	$k_4 \oplus t_3$	$k_5 \oplus t_4$	$k_6 \oplus t_5$	$k_7 \oplus t_6$	$k_1 \oplus t_7$	$k_2 \oplus t_8$
13	$k_3 \oplus t_1$	$k_4 \oplus t_2$	$k_5 \oplus t_3$	$k_6 \oplus t_4$	$k_7 \oplus t_5$	$k_1 \oplus t_6$
14	$k_2 \oplus t_7$	$k_3 \oplus t_8$	$k_4 \oplus t_1$	$k_5 \oplus t_2$	$k_6 \oplus t_3$	$k_7 \oplus t_4$
15	$k_1 \oplus t_5$	$k_2 \oplus t_6$	$k_3 \oplus t_7$	$k_4 \oplus t_8$	$k_5 \oplus t_1$	$k_6 \oplus t_2$
16	$k_7 \oplus t_3$	$k_1 \oplus t_4$	$k_2 \oplus t_5$	$k_3 \oplus t_6$	$k_4 \oplus t_7$	$k_5 \oplus t_8$

3.5 S-box Properties and Probable Generation and Selection Criteria

The facts that none of the available descriptions of the algorithm mention anything about the S-box selection criteria and that the S-box is the only nonlinear part of the cipher make its properties important to study. This has been done using the techniques described in [26]. Following the proposed strategy in that article, the S-box was first studied using what the authors call the "Pollock" technique. The name alludes to the 20th century abstract expressionist painter and simply consists of plotting the S-box's LAT and DDT, studying them to find non-random patterns. The visualizations of the LAT and DDT are shown in Figures 3.3 and 3.4, respectively. Inspection of them does not reveal any obvious non-random patterns.

In [27], the study of the visual representation of the LAT modulo 4 is suggested. It notes that the presence of patterns there can indicate that the S-box was generated by a Feistel network with a low number of rounds. The LAT modulo 4 of the SoDark S-box is shown in Figure 3.5. It does indeed show unmistakable patterns. For that reason, the possibility that the S-box was generated by a Feistel network was investigated using the techniques described in [26]. Algorithm 2 from that article, DecomposeFeistel, was implemented to generate a CNF representation of a Feistel network that can generate the S-box. This representation was then used as input to the SAT solvers CryptoMiniSat [28] and Treengeling [29], which found the problem unsatisfiable. This ruled out the possibility that the S-box was generated by a Feistel network with bijective round functions and five or fewer rounds. The authors of [27] have noted in an associated presentation, that randomly generated S-boxes can have patterns in the LAT modulo 4 that look similar to those in S-boxes generated by Feistel networks with more than five rounds.

With the hypothesis that the S-box was generated by a low round Feistel network falsified, the possibility that the S-box was a randomly selected permutation was investigated. In [26], the probability distribution of the coefficients in the LAT of a random permutation is given as

$$P\left[c_{i,j} = 2z\right] = \frac{\binom{2^{n-1}}{2^{n-2}+z}^2}{\binom{2^n}{2^{n-1}}} \tag{3.19}$$

where $P\left[c_{i,j} = 2z\right]$ is the probability that a particular combination of input and output bits will have the bias $2z$ and n is the S-box width in bits. This probability distribution is plotted together with the distribution of the SoDark S-box LAT in Figures 3.6 and 3.7. The predicted and actual distributions track each other very closely and a χ^2 test was made to establish the goodness of fit. With $\chi^2 = 91.3$ and 38 degrees of freedom, this yields a p-value less than 0.00001, which indicates a very high likelihood that the SoDark S-box was chosen randomly. The only selection criteria was probably that there could be no fixed points, i.e., no number $X \in \{0, 1\}^8$ such that $f(X) = X$.

It should be noted that the χ^2 test assumes that each trial in the experiment is independent of the other trials. This is not strictly true in the case of the different factors in a LAT. The χ^2 measure is still used here though, since it is believed to be a good approximation of the goodness of fit, despite non-independence of the LAT biases.

The fact that the S-box was chosen at random means that it is unlikely to have the properties that are considered important for S-boxes used in modern ciphers. In particular, randomly chosen S-boxes are typically vulnerable to both linear and differential cryptanalysis [16]. That this is the case here can be understood by studying Figures 3.3, 3.4, 3.6, and 3.7. The highest linear bias is $\frac{38}{256}$, slightly higher than the average expected bias of a random permutation (see Figure 3.7). In regard to resistance to differential cryptanalysis, the delta uniformity (highest value in the DDT) is also high at 14. This can be put in contrast to the delta uniformity of S-boxes that have been engineered to provide resistance to differential cryptanalysis, such as the AES S-box, where the delta uniformity is 4. The large number of high probability differentials in the DDT also means that it has a large number of differentials with probability zero.

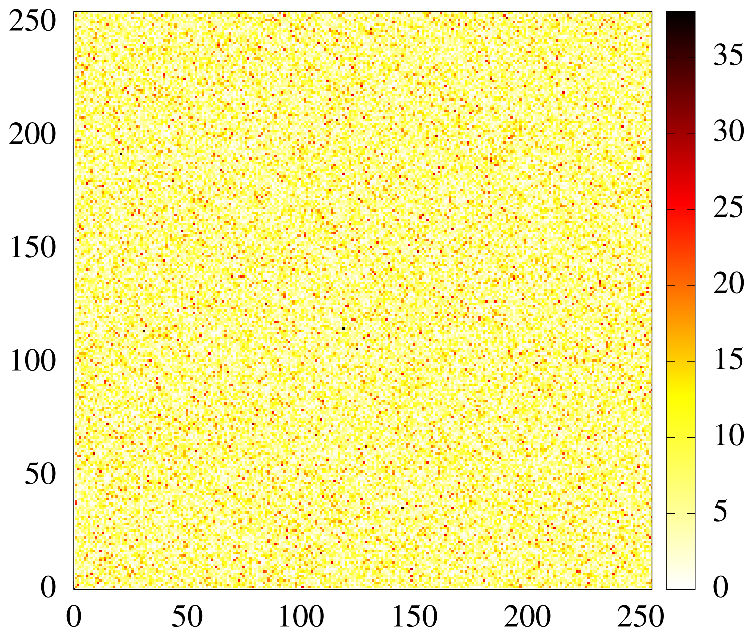

Figure 3.3. Graphic representation of the SoDark S-box LAT.

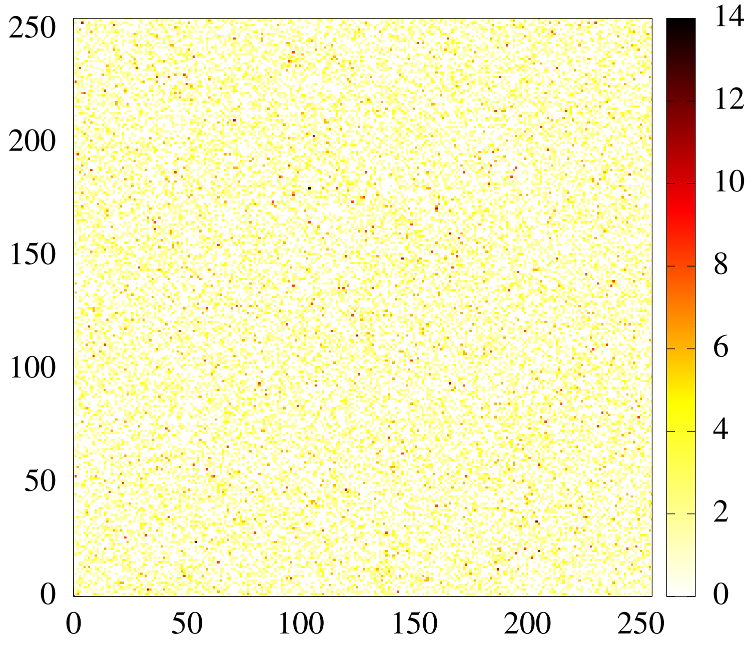

Figure 3.4. Graphic representation of the SoDark S-box DDT.

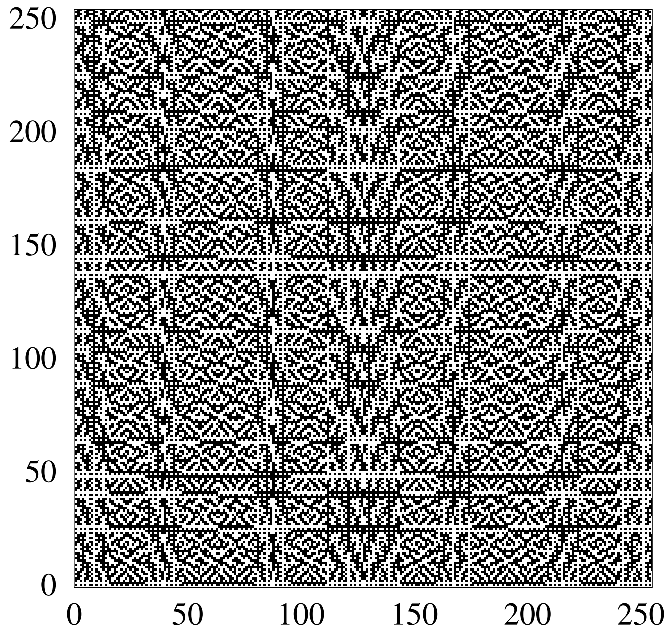

Figure 3.5. Graphic representation of the SoDark S-box DDT modulo 4.

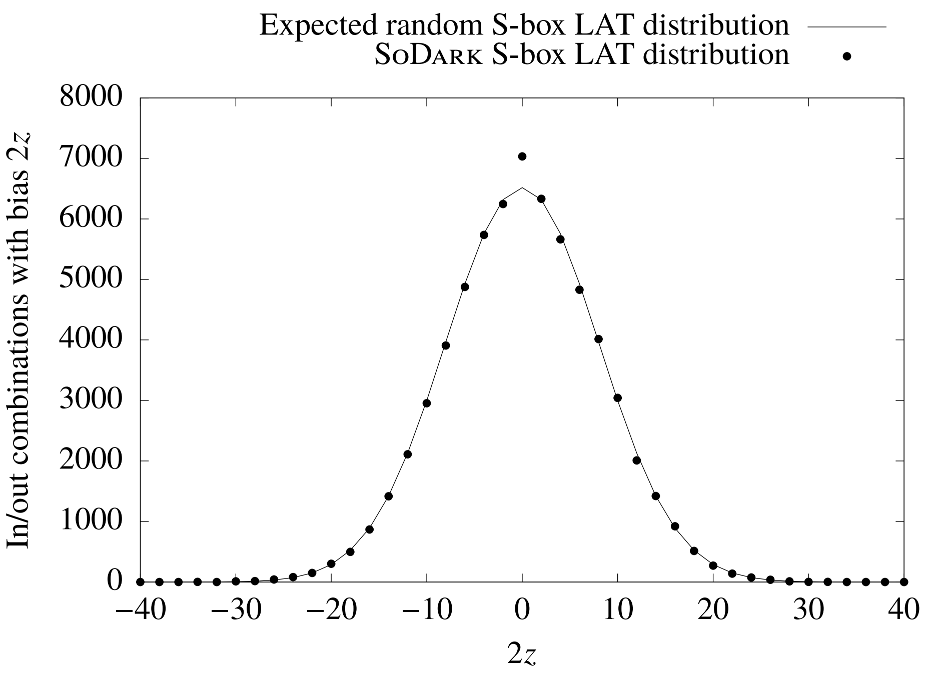

Figure 3.6. Linear approximation distribution.

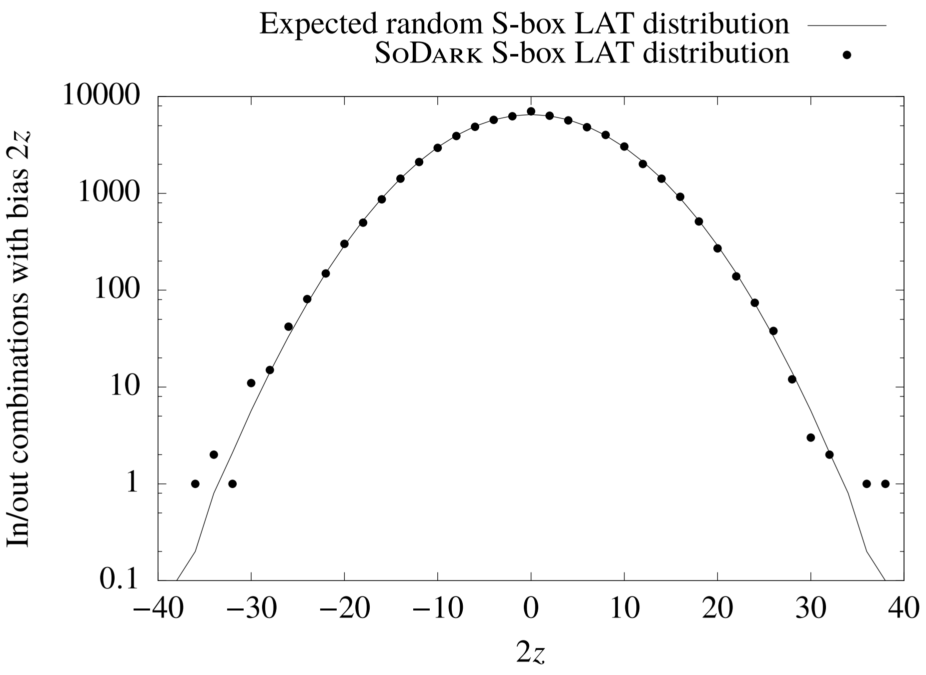

Figure 3.7. Linear approximation distribution, logarithmic scale.

3.6 Equivalence to the Even–Mansour Construction

Due to the commutative property of the XOR operation, each round of the algorithm can be rewritten as a function of one 24-bit input vector $P_r = A \parallel B \parallel C$:

$$P_{r+1} = g(P_r \oplus K_r) \tag{3.20}$$

where g is a bijective mapping $g : \{0,1\}^{24} \longrightarrow \{0,1\}^{24}$ defined as

$$g(X) = g(A \parallel B \parallel C) = s(A) \oplus B' \parallel B' \parallel s(C) \oplus B' \tag{3.21}$$

and

$$B' = s\left(s(A) \oplus B \oplus s(C)\right). \tag{3.22}$$

The transformation

$$T(X) = T(A \parallel B \parallel C) = A \oplus B \parallel B \parallel B \oplus C \tag{3.23}$$

must also be applied before the first and after the last round to ensure the rewritten algorithm is equivalent to the original definition. It follows from the definition of g that it is bijective, provided that s is bijective. The SoDark algorithm with r rounds can now be expressed as

$$E_K(\mathcal{P}) = T\left(g\left(g\left(g\left(g\left(T\left(\mathcal{P}\right) \oplus K_1\right) \oplus K_2\right) \ \ldots \ \oplus K_{r-1}\right) \oplus K_r\right)\right) \tag{3.24}$$

where $K_r = k_1 \parallel k_3 \parallel k_2$ with values from in Table 3.2. Figure 3.8 shows the algorithm expressed in this manner. Decryption is identical to encryption with g^{-1} in place of g and the round keys applied in reverse order. A representation of SoDark-6 can be derived in the same manner.

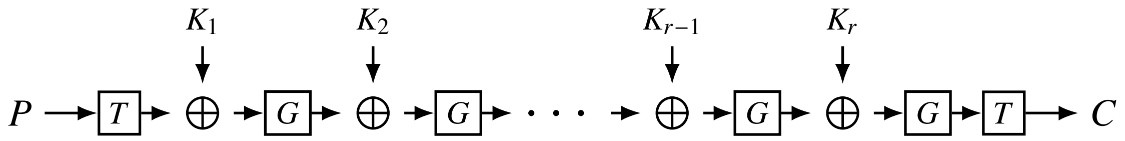

Figure 3.8. An r-round iterated Even–Mansour construction with round function G and initial and final transformations T.

From Equation 3.24, it now clear that the algorithm is equivalent to the iterated EM construction [30], with g as the random permutation function and the transformation T applied to the plaintext and ciphertext. The applications of T and the last application of g provide no additional security as their inverses are known.

3.7 Properties with Respect to Linear Cryptanalysis

Since the 8-bit S-box had a number of linear combinations of input and output bits with high bias, the assumption was made that the prevalence of high-bias linearities would remain in the transformation into a 24-bit S-box. It is not feasible to generate the full LAT for a 24-bit S-box, since this process has very high time and memory complexities. For that reason, only a part of the set of possible linearizations has been searched.

Initially, all combinations of one, two, three, and four input and output bits were searched to find good linearizations. This yielded a number of linearizations with significant bias—some over 10%. The best linearizations found using this method are presented in Table 3.4.

In order to find more high-bias linearizations of the 24-bit S-box, a heuristic search algorithm was used. Different combinations of high-linearity input masks for the 8-bit S-box and their corresponding output masks were tried on the 24-bit S-box. The results of this were surprisingly good: Linearizations with up to 14.8% bias were found. The best known linearizations for the 24-bit S-box are presented in Table 3.5.

In all, 111 linearizations with a bias of more than 10% have been found.

Using a branch and bound algorithm, combinations of the S-box linearizations that approximate five rounds of the cipher were found, i.e., the number of rounds needed for an attack of the eight-round variant used in 2G ALE. The biases of those linearizations are so low that even if given all 2^{24} theoretically possible plaintext messages and their corresponding ciphertexts, the probability of recovering key bits faster than brute force is prohibitively high.

Table 3.4. Linearizations of the 24-bit SoDark S-box found by searching all one-, two-, three-, and four-bit combinations.

Input mask	Output mask	Bias
`000060`	`002222`	−10.94%
`600000`	`222200`	−10.94%
`0000C8`	`00A0A0`	−10.94%
`C80000`	`A0A000`	−10.94%
`00009A`	`000202`	−10.94%
`9A0000`	`202000`	−10.94%

Table 3.5. Best known linearizations for the 24-bit SoDark algorithm S-box.

Input mask	Output mask	Bias
`000073`	`007777`	14.8%
`730000`	`777700`	14.8%
`000024`	`009191`	14.1%
`240000`	`919100`	14.1%
`0000C0`	`001515`	−14.1%
`C00000`	`151500`	−14.1%

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Structural Attacks

4.1 Measures of Complexity

The efficiency of a cryptographic attack is measured by its complexity. It provides a means of relating the speed of the attack to that of a brute force approach, or to compare different attacks with each other. An attack's complexity can be stated for its time, data, and memory requirements. Time complexity specifies how many operations of some kind that the attack requires on average. It is normally the most important complexity considered. Data complexity specifies the amount of data needed in the form of plaintext–ciphertext pairs, or the like, to perform the attack. Lastly, an attack's memory complexity describes the amount of memory that it needs to run.

For the attacks presented in this and following chapters, complexities are stated in exponential notation. In the case of an attack on r rounds, the unit used to describe the time complexity is the number of r-round encryptions that would take the same time to perform. As an example, a brute force attack on SoDark—which uses 56-bit keys—is expected to have a complexity of 2^{55} on average.

The speed of SoDark implementations is almost entirely dependent on the number of S-box operations performed. The number and speed of all other operations required in encryption, decryption, and attacks are negligible in comparison. For that reason, the number of S-box operations required to test one key in a brute force attack will be used to calculate the relative time complexity of other attacks. Testing one key in a r-round brute force attack requires $3 \cdot (r - 1)$ S-box operations.

For data complexities, the unit used is the number of known ciphertexts, plaintext–ciphertext pairs, or plaintext–ciphertext–tweak tuples that are needed to perform the attack with some specified probability of success.

For memory complexities, the unit is the cipher block size.

For time complexities in particular, the actual complexity required to perform an attack may vary. This is both because of the "luck" aspect—the correct key could be among the first or last tried—but also because the number of matching differentials in some intermediate state of the cipher may be unusually high or low. In the calculations of time complexities, the average number of one-byte keys implying a specific one-byte target differential was used. This number is believed to result in the best estimates of average-case performance. It was calculated from the SoDark S-box DDT as $\frac{65280}{25544} = \frac{8160}{3193} \approx 2.6$. Over the set of all possible one-byte keys, the average number of possible output differentials for a given input differential to the S-box is $\frac{25544}{255} \approx 100$. Both these averages exclude the zero differential, which only implies itself.

4.2 Attacks on Iterated Even–Mansour Constructions

It is immediately apparent that one round of SoDark provides no security at all since, given one plaintext–ciphertext–tweak tuple $(\mathcal{P}, \mathcal{C}, \mathcal{T})$, the key can be recovered by $\mathcal{K} = g^{-1}(\mathcal{C}) \oplus \mathcal{P} \oplus \mathcal{T}$. Two rounds of the algorithm is equivalent to the original one-round EM construction described in [30].

Known and chosen plaintext attacks on the EM construction corresponding to the lower bound proven by Even and Mansour in [30] are presented by Daemen in [31]. For the case of two independent subkeys of size n, Daemen shows a known-plaintext attack with an average time complexity proportional to 2^{n-1} and a chosen plaintext attack with complexity proportional to $2^{\frac{n}{2}}$. Both of these are significantly faster than the 2^{2n} complexity of a brute force attack. Thus, the two-round SoDark-3 algorithm provides at most 12 bits of security in regard to this attack. Further insight is given by [32], which shows that independent subkeys in the single-round EM construction provide no added security compared to a construction with identical subkeys.

Attacks on various iterated versions of the EM construction are presented in [33], [34], and [35]. Notably, [34] demonstrates that, for ≤ 4 rounds with two independent keys used in any order throughout the rounds, the time complexity for recovering the keys is at most proportional to 2^n. (An r-round iterated EM construction uses $r + 1$ keys.)

Generalizing, [36] shows that the an r-round iterated EM construction with independent round keys has an upper security bound of $r \cdot 2^{\frac{rn}{r+1}}$ queries to an oracle, where n is the block size. The article also shows an attack for an r-round iterated EM construction with time complexity proportional to $2^{\frac{rn}{2}}$.

While attacks on the SoDark cipher that consider it as an EM construction are directly applicable, they are suboptimal because they regard the 24-bit S-box as a random permutation. In reality, it is a combination of three 8-bit S-boxes (see Equations 3.21 and 3.22). This structure can be used to mount the more efficient attacks described in the following sections.

4.3 Known-Plaintext Attack on Two-Round SoDark-3

The calculations for two rounds of encryption using SoDark-3 are:

$$A^{(1)} = s\Big(A^{(0)} \oplus B^{(0)} \oplus k_1 \oplus t_1\Big) \tag{4.1}$$

$$C^{(1)} = s\Big(C^{(0)} \oplus B^{(0)} \oplus k_2 \oplus t_2\Big) \tag{4.2}$$

$$B^{(1)} = s\Big(B^{(0)} \oplus A^{(1)} \oplus C^{(1)} \oplus k_3 \oplus t_3\Big) \tag{4.3}$$

$$A^{(2)} = s\Big(A^{(1)} \oplus B^{(1)} \oplus k_4 \oplus t_4\Big) \tag{4.4}$$

$$C^{(2)} = s\Big(C^{(1)} \oplus B^{(1)} \oplus k_5 \oplus t_5\Big) \tag{4.5}$$

$$B^{(2)} = s\Big(B^{(1)} \oplus A^{(2)} \oplus C^{(2)} \oplus k_6 \oplus t_6\Big). \tag{4.6}$$

Since the inverse s^{-1} and tweak is known

$$\hat{B}^{(2)} = s^{-1}\Big(B^{(2)}\Big) \oplus A^{(2)} \oplus C^{(2)} \oplus t_6 = B^{(1)} \oplus k_6 \tag{4.7}$$

$$\hat{C}^{(2)} = s^{-1}\Big(C^{(2)}\Big) \oplus t_5 = C^{(1)} \oplus B^{(1)} \oplus k_5 \tag{4.8}$$

$$\hat{A}^{(2)} = s^{-1}\Big(A^{(2)}\Big) \oplus t_4 = A^{(1)} \oplus B^{(1)} \oplus k_4 \tag{4.9}$$

can be calculated. From Equations 4.1 through 4.6, it is also evident that

$$\hat{A}^{(2)} = s\Big(A^{(0)} \oplus B^{(0)} \oplus k_1 \oplus t_1\Big) \oplus s\Big(B^{(0)} \oplus A^{(1)} \oplus C^{(1)} \oplus k_3 \oplus t_3\Big) \oplus k_4 \tag{4.10}$$

and

$$\hat{C}^{(2)} = s\Big(C^{(0)} \oplus B^{(0)} \oplus k_2 \oplus t_2\Big) \oplus s\Big(B^{(0)} \oplus A^{(1)} \oplus C^{(1)} \oplus k_3 \oplus t_3\Big) \oplus k_5. \tag{4.11}$$

Now, given two plaintext–ciphertext–tweak tuples, the differentials $\Delta A^{(1)}$, $\Delta C^{(1)}$, and $\Delta B^{(0)}$ can be calculated:

$$\begin{aligned}
\Delta A^{(1)} &= \hat{A}_1^{(2)} \oplus \hat{A}_2^{(2)} \oplus \hat{B}_1^{(2)} \oplus \hat{B}_2^{(2)} \\
&= s\Big(A_1^{(0)} \oplus B_1^{(0)} \oplus k_1 \oplus t_{1,1}\Big) \oplus s\Big(B_1^{(0)} \oplus A_1^{(1)} \oplus C_1^{(1)} \oplus k_3 \oplus t_{3,1}\Big) \oplus \\
&\quad s\Big(A_2^{(0)} \oplus B_2^{(0)} \oplus k_1 \oplus t_{1,2}\Big) \oplus s\Big(B_2^{(0)} \oplus A_2^{(1)} \oplus C_2^{(1)} \oplus k_3 \oplus t_{3,2}\Big) \oplus \\
&\quad s\Big(B_1^{(0)} \oplus A_1^{(1)} \oplus C_1^{(1)} \oplus k_3 \oplus t_{3,1}\Big) \oplus s\Big(B_2^{(0)} \oplus A_2^{(1)} \oplus C_2^{(1)} \oplus k_3 \oplus t_{3,2}\Big) \\
&= s\Big(A_1^{(0)} \oplus B_1^{(0)} \oplus k_1 \oplus t_{1,1}\Big) \oplus s\Big(A_2^{(0)} \oplus B_2^{(0)} \oplus k_1 \oplus t_{1,2}\Big) \\
&= A_1^{(1)} \oplus A_2^{(1)}
\end{aligned} \tag{4.12}$$

$$\begin{aligned}
\Delta C^{(1)} &= \hat{C}_1^{(2)} \oplus \hat{C}_2^{(2)} \oplus \hat{B}_1^{(2)} \oplus \hat{B}_2^{(2)} \\
&= s\Big(C_1^{(0)} \oplus B_1^{(0)} \oplus k_2 \oplus t_{2,1}\Big) \oplus s\Big(B_1^{(0)} \oplus A_1^{(1)} \oplus C_1^{(1)} \oplus k_3 \oplus t_{3,1}\Big) \oplus \\
&\quad s\Big(C_2^{(0)} \oplus B_2^{(0)} \oplus k_2 \oplus t_{2,2}\Big) \oplus s\Big(B_2^{(0)} \oplus A_2^{(1)} \oplus C_2^{(1)} \oplus k_3 \oplus t_{3,2}\Big) \oplus \\
&\quad s\Big(B_1^{(0)} \oplus A_1^{(1)} \oplus C_1^{(1)} \oplus k_3 \oplus t_{3,1}\Big) \oplus s\Big(B_2^{(0)} \oplus A_2^{(1)} \oplus C_2^{(1)} \oplus k_3 \oplus t_{3,2}\Big) \\
&= s\Big(C_1^{(0)} \oplus B_1^{(0)} \oplus k_2 \oplus t_{2,1}\Big) \oplus s\Big(C_2^{(0)} \oplus B_2^{(0)} \oplus k_2 \oplus t_{2,2}\Big) \\
&= C_1^{(1)} \oplus C_2^{(1)}
\end{aligned} \tag{4.13}$$

$$\Delta B^{(0)} = s^{-1}\left(\hat{B}_1^{(2)} \oplus k_6\right) \oplus s^{-1}\left(\hat{B}_2^{(2)} \oplus k_6\right) \oplus \Delta A^{(1)} \oplus \Delta C^{(1)} \oplus t_{3,1} \oplus t_{3,2}. \tag{4.14}$$

Equations 4.12, 4.13, and 4.14 show that the candidates for key bytes k_1 and k_2, and k_6 can be searched independently of each other and the other key bytes. Each value of k_1, k_2, and k_6 will imply a value for $\Delta A^{(1)}$, $\Delta C^{(1)}$, and $\Delta B^{(0)}$, respectively, and those that do not generate the differential calculated from the ciphertext—or the plaintext in the case of k_6—can be immediately discarded. This process is shown in Figure 4.1.

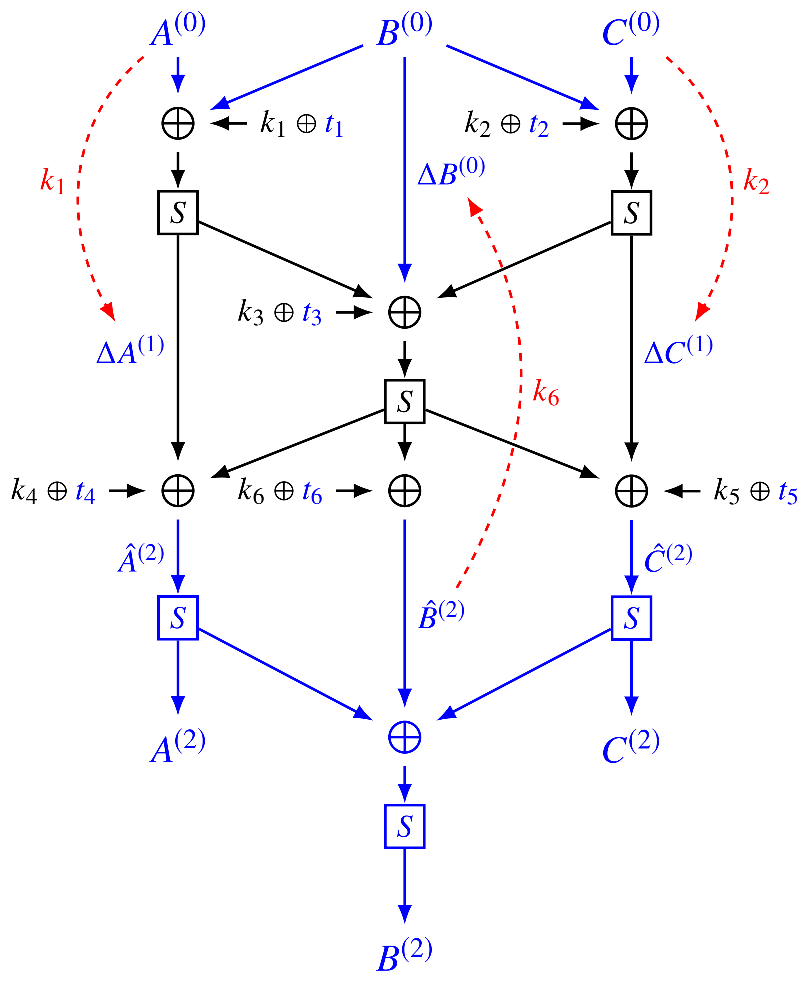

Figure 4.1. Attack on two-round SoDark-3 by guessing key bytes k_1, k_2, and k_6 independently and matching the results with $\Delta A^{(1)}$, $\Delta C^{(1)}$, and $\Delta B^{(0)}$. The parts of the cipher marked in blue are known or can be calculated without guessing any part of the key.

On average, 2.6 candidate values each for k_1, k_2, and k_6 are expected as a result of this search. Now, for each possible tuple k_1, k_2, k_6, the values of k_3, k_4, k_5 are calculated. If the values of those match for both plaintext–ciphertext–tweak tuples, we have a candidate key that can be verified against further plaintext–ciphertext–tweak tuples. The full attack process is described in Algorithm 4.1.

In calculating the time complexity of the attack, we first note that 2^8 keys have to be tested for each of k_1, k_2, and k_6. Each test uses two S-box operations. This will yield, on average, $2.6^3 \approx 17.6$ candidate k_1, k_2, k_6 tuples. For each of those, k_3, k_4, and k_5 are calculated using both plaintext–ciphertext–tweak tuples. This requires six S-box operations per key tuple, but some of those can be cached in between iterations, see Algorithm 4.1. Therefore, the total average time complexity of the two-round attack is

$$\frac{6 \cdot 2^8 + 2 \cdot 2.6 + 2 \cdot 2.6^2 + 2 \cdot 2.6^3}{3} \approx 2^9. \tag{4.15}$$

Any pair of plaintext–ciphertext–tweak tuples that satisfy

$$\Delta A^{(0)} \oplus \Delta B^{(0)} \oplus t_{1,1} \oplus t_{1,2} \neq 0 \tag{4.16}$$

$$\Delta C^{(0)} \oplus \Delta B^{(0)} \oplus t_{2,1} \oplus t_{2,2} \neq 0 \tag{4.17}$$

$$\Delta A^{(1)} \oplus \Delta B^{(0)} \oplus \Delta C^{(1)} \oplus t_{3,1} \oplus t_{3,2} \neq 0 \tag{4.18}$$

can be used in attack. Since the number of tuple pairs that does not satisfy this requirement is quite small, the attack works for virtually any pair, making the data complexity 2. The memory complexity is $2^{4.1}$.

Algorithm 4.1 Perform a known-plaintext attack on two-round SoDark-3 and print all candidate keys.

```
1:  procedure CrackTwoRounds($\mathcal{P}_1, \mathcal{C}_1, \mathcal{T}_1, \mathcal{P}_2, \mathcal{C}_2, \mathcal{T}_2$)
2:      $\Delta B^{(0)} \leftarrow B_1^{(0)} \oplus B_2^{(0)}$
3:      $\hat{B}_1^{(2)} \leftarrow s^{-1}(B_1^{(2)}) \oplus A_1^{(2)} \oplus C_1^{(2)} \oplus t_{6,1}$
4:      $\hat{B}_2^{(2)} \leftarrow s^{-1}(B_2^{(2)}) \oplus A_2^{(2)} \oplus C_2^{(2)} \oplus t_{6,2}$
5:      $\Delta B^{(1)} \leftarrow \hat{B}_1^{(1)} \oplus \hat{B}_2^{(1)}$
6:      $\hat{A}_1^{(2)} \leftarrow s^{-1}(A_1^{(2)}) \oplus t_{4,1}$
7:      $\hat{A}_2^{(2)} \leftarrow s^{-1}(A_2^{(2)}) \oplus t_{4,2}$
8:      $\Delta A^{(1)} \leftarrow \hat{A}_1^{(2)} \oplus \hat{A}_2^{(2)} \oplus \Delta B^{(1)}$
9:      $\hat{C}_1^{(2)} \leftarrow s^{-1}(C_1^{(2)}) \oplus t_{5,1}$
10:     $\hat{C}_2^{(2)} \leftarrow s^{-1}(C_2^{(2)}) \oplus t_{5,2}$
11:     $\Delta C^{(1)} \leftarrow \hat{C}_1^{(2)} \oplus \hat{C}_2^{(2)} \oplus \Delta B^{(1)}$
12:     $L_{k_1} \leftarrow$ empty list
13:     $L_{k_2} \leftarrow$ empty list
14:     $L_{k_6} \leftarrow$ empty list
15:     for all $k_1$ do                                          ▷ $2^8$ possible $k_1$
16:         if $s(A_1^{(0)} \oplus B_1^{(0)} \oplus k_1 \oplus t_{1,1}) \oplus s(A_2^{(0)} \oplus B_2^{(0)} \oplus k_1 \oplus t_{1,2}) = \Delta A^{(1)}$ then
17:             $L_{k_1}$.append($k_1$)
18:         end if
19:     end for
20:     for all $k_2$ do                                          ▷ $2^8$ possible $k_2$
21:         if $s(C_1^{(0)} \oplus B_1^{(0)} \oplus k_2 \oplus t_{2,1}) \oplus s(C_2^{(0)} \oplus B_2^{(0)} \oplus k_2 \oplus t_{2,2}) = \Delta C^{(1)}$ then
22:             $L_{k_2}$.append($k_2$)
23:         end if
24:     end for
25:     for all $k_6$ do                                          ▷ $2^8$ possible $k_6$
26:         if $s^{-1}(\hat{B}_1^{(2)} \oplus k_6) \oplus s^{-1}(\hat{B}_2^{(2)} \oplus k_6) \oplus \Delta A^{(1)} \oplus \Delta C^{(1)} \oplus t_{3,1} \oplus t_{3,2} = \Delta B^{(0)}$ then
27:             $L_{k_6}$.append($k_6$)
28:         end if
29:     end for
```

```
30:     for all $k_1 \in L_{k_1}$ do                                          ▷ Average 2.6 $k_1$
31:         $A_1^{(1)} \leftarrow s(A_1^{(0)} \oplus B_1^{(0)} \oplus k_1 \oplus t_{1,1})$
32:         $A_2^{(1)} \leftarrow s(A_2^{(0)} \oplus B_2^{(0)} \oplus k_1 \oplus t_{1,2})$
33:         for all $k_2 \in L_{k_2}$ do                                      ▷ Average 2.6 $k_2$
34:             $C_1^{(1)} \leftarrow s(C_1^{(0)} \oplus B_1 \oplus k_2 \oplus t_{2,1})$
35:             $C_2^{(1)} \leftarrow s(C_2^{(0)} \oplus B_2 \oplus k_2 \oplus t_{2,2})$
36:             for all $k_6 \in L_{k_6}$ do                                  ▷ Average 2.6 $k_6$
37:                 $B_1^{(1)} \leftarrow \hat{B}_1^{(2)} \oplus k_6$
38:                 $B_2^{(1)} \leftarrow \hat{B}_2^{(2)} \oplus k_6$
39:                 $k_{3,1} \leftarrow B_1^{(0)} \oplus A_1^{(1)} \oplus C_1^{(1)} \oplus s^{-1}(B_1^{(1)}) \oplus t_{3,1}$
40:                 $k_{3,2} \leftarrow B_2^{(0)} \oplus A_2^{(1)} \oplus C_2^{(1)} \oplus s^{-1}(B_2^{(1)}) \oplus t_{3,2}$
41:                 $k_{4,1} \leftarrow B_1^{(1)} \oplus A_1^{(1)} \oplus \hat{A}_1^{(2)}$
42:                 $k_{4,2} \leftarrow B_2^{(1)} \oplus A_2^{(1)} \oplus \hat{A}_2^{(2)}$
43:                 $k_{5,1} \leftarrow B_1^{(1)} \oplus C_1^{(1)} \oplus \hat{C}_1^{(2)}$
44:                 $k_{5,2} \leftarrow B_2^{(1)} \oplus C_2^{(1)} \oplus \hat{C}_1^{(2)}$
45:                 if $k_{3,1} = k_{3,2}$ and $k_{4,1} = k_{4,2}$ and $k_{5,1} = k_{5,2}$ then
46:                     $k_3 \leftarrow k_{3,1}$
47:                     $k_4 \leftarrow k_{4,1}$
48:                     $k_5 \leftarrow k_{5,1}$
49:                     PRINT($k_1 \parallel k_2 \parallel k_3 \parallel k_4 \parallel k_5 \parallel k_6$)
50:                 end if
51:             end for
52:         end for
53:     end for
54: end procedure
```

4.4 Known-Plaintext Attack on Three-Round SoDark-3

The attack on three-round SoDark-3 is a direct extension of the two-round attack described in the previous section. The encryption process is analogous to the one shown in Equations 4.1 through 4.6 and the last round can be partially reversed to calculate $\hat{A}^{(3)}$, $\hat{B}^{(3)}$, and $\hat{C}^{(3)}$ using the method shown in Equations 4.7 through 4.9.

The attack is shown graphically in Figure 4.2. It uses the fact that two of the three bytes in the first and last round keys are identical to perform partial differential matching in the middle round.

First, by guessing key byte k_2, $\Delta C^{(1)}$ can be calculated from the plaintext as

$$C_1^{(1)} = s\left(C_1^{(0)} \oplus B_1^{(0)} \oplus k_2 \oplus t_{2,1}\right) \tag{4.19}$$

$$C_2^{(1)} = s\left(C_2^{(0)} \oplus B_2^{(0)} \oplus k_2 \oplus t_{2,2}\right) \tag{4.20}$$

$$\Delta C^{(1)} = C_1^{(1)} \oplus C_2^{(1)} \tag{4.21}$$

and, by calculating $\Delta A^{(2)}$ and $\Delta C^{(2)}$ in the same way as in Equations 4.12 and 4.13, $\Delta B^{(1)}$ can be calculated from the ciphertext as

$$\Delta B^{(1)} = \Delta A^{(2)} \oplus \Delta C^{(2)} \oplus s^{-1}\left(\hat{B}_1^{(3)} \oplus k_2\right) \oplus s^{-1}\left(\hat{B}_2^{(3)} \oplus k_2\right) \oplus t_{6,1} \oplus t_{6,2}. \tag{4.22}$$

Now, the value of $\Delta C^{(1)}$ can be compared with $\Delta B^{(1)} \oplus \Delta\hat{C}^{(2)}$, where $\Delta\hat{C}^{(2)}$ is calculated by guessing k_1 in addition to k_2:

$$B_1^{(2)} = \hat{B}_1^{(3)} \oplus k_2 \tag{4.23}$$

$$B_2^{(2)} = \hat{B}_2^{(3)} \oplus k_2 \tag{4.24}$$

$$C_1^{(2)} = B_1^{(2)} \oplus \hat{C}_1^{(3)} \oplus k_1 \tag{4.25}$$

$$C_2^{(2)} = B_2^{(2)} \oplus \hat{C}_2^{(3)} \oplus k_1 \tag{4.26}$$

$$\Delta\hat{C}^{(2)} = s^{-1}\left(C_1^{(2)}\right) \oplus s^{-1}\left(C_2^{(2)}\right) \oplus t_{5,1} \oplus t_{5,2}. \tag{4.27}$$

If $\Delta C^{(1)}$ and $\Delta B^{(1)} \oplus \Delta\hat{C}^{(2)}$ are equal, the k_1, k_2 pair is a candidate for those key bytes. This is expected to happen with probability $\frac{1}{256}$, resulting in 2^8 candidates for k_1, k_2.

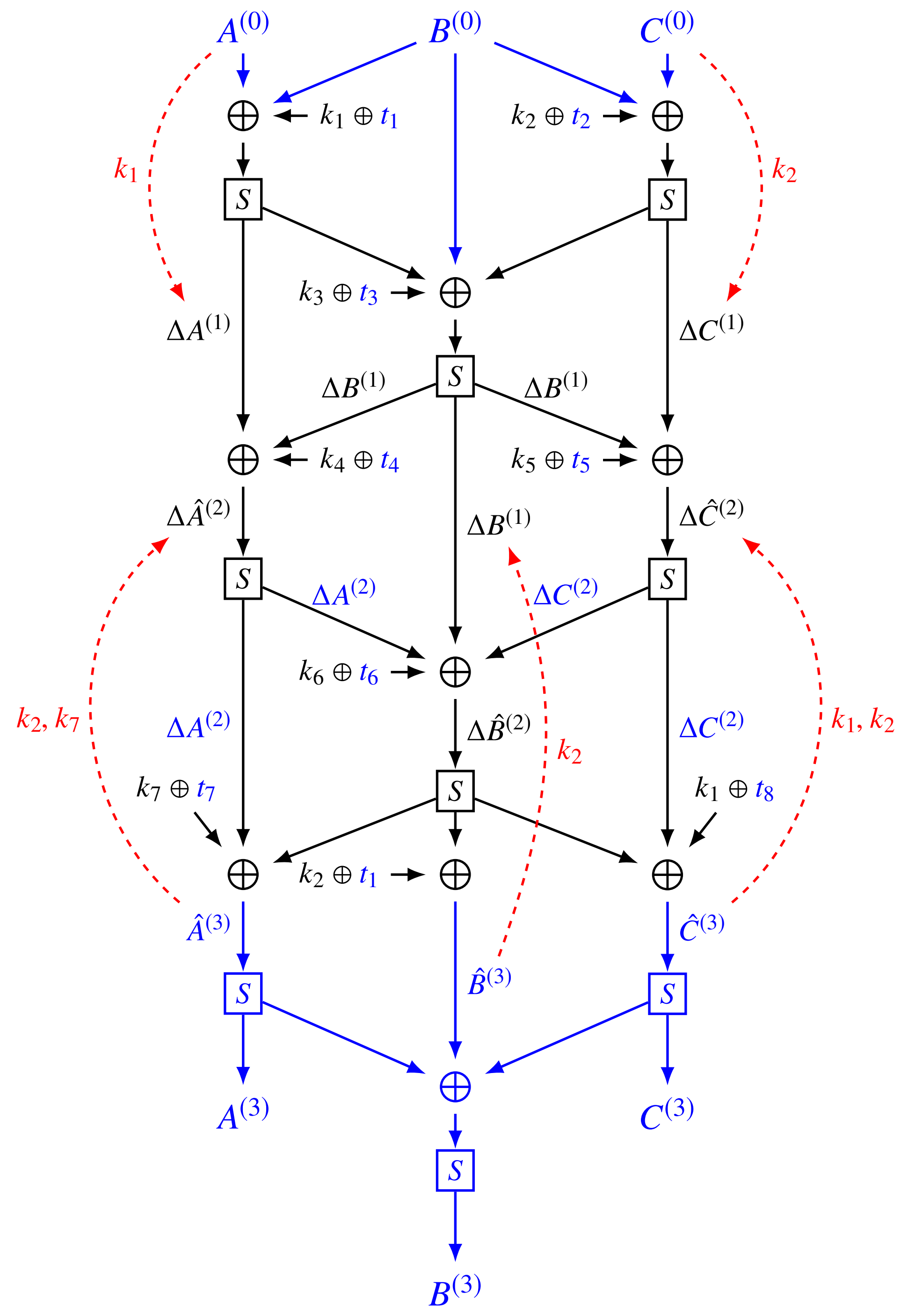

Figure 4.2. Attack on three-round SoDark-3 by first guessing key bytes k_1 and k_2 independently and matching on $\Delta C^{(1)}$. In the case of that match, k_7 is guessed and matching on $\Delta A^{(1)}$ is performed. The parts of the cipher marked in blue are known or can be calculated without guessing any part of the key.

For each candidate pair k_1, k_2, possible values for k_7 are then sought. This is done by guessing k_7 and comparing $\Delta A^{(1)}$ to $\Delta \hat{A}^{(2)} \oplus \Delta B^{(2)}$:

$$
\begin{aligned}
A_1^{(1)} &= s\left(A_1^{(0)} \oplus B_1^{(0)} \oplus k_1 \oplus t_{1,1}\right) && (4.28)\\
A_2^{(1)} &= s\left(A_2^{(0)} \oplus B_2^{(0)} \oplus k_1 \oplus t_{1,2}\right) && (4.29)\\
\Delta A^{(1)} &= A_1^{(1)} \oplus A_2^{(1)} && (4.30)\\
A_1^{(2)} &= \hat{A}_1^{(3)} \oplus B_1^{(2)} \oplus k_7 && (4.31)\\
A_2^{(2)} &= \hat{A}_2^{(3)} \oplus B_2^{(2)} \oplus k_7 && (4.32)\\
\Delta \hat{A}^{(2)} &= s^{-1}\left(A_1^{(2)}\right) \oplus s^{-1}\left(A_2^{(2)}\right) \oplus t_{4,1} \oplus t_{4,2}. && (4.33)
\end{aligned}
$$

As before, if $\Delta A^{(1)}$ and $\Delta B^{(1)} \oplus \Delta \hat{A}^{(2)}$ match, then the tuple k_1, k_2, k_7 is a candidate for those key bytes. For the same reason, we expect 2^8 candidates for k_1, k_2, k_7 to remain after this step.

Finally, for each candidate tuple k_1, k_2, k_7, possible values of k_3 are found by checking that the value of k_4 implied by a guessed k_3 is the same for both plaintext–ciphertext–tweak tuples:

$$
\begin{aligned}
k_{4,1} &= s\left(A_1^{(1)} \oplus C_1^{(1)} \oplus B_1^{(0)} \oplus k_3 \oplus t_{3,1}\right) \oplus s^{-1}\left(\hat{A}_1^{(3)} \oplus B_1^{(2)} \oplus k_7\right) \oplus A_1^{(1)} \oplus t_{4,1} && (4.34)\\
k_{4,2} &= s\left(A_2^{(1)} \oplus C_2^{(1)} \oplus B_2^{(0)} \oplus k_3 \oplus t_{3,2}\right) \oplus s^{-1}\left(\hat{A}_2^{(3)} \oplus B_2^{(2)} \oplus k_7\right) \oplus A_2^{(1)} \oplus t_{4,2}. && (4.35)
\end{aligned}
$$

Then, the values of k_5 and k_6 can be calculated from the values already known, thus yielding a full candidate key:

$$
\begin{aligned}
k_5 &= s^{-1}\left(C_1^{(2)}\right) \oplus B_1^{(1)} \oplus C_1^{(1)} \oplus t_{5,1} && (4.36)\\
k_6 &= s^{-1}\left(B_1^{(2)}\right) \oplus A_1^{(2)} \oplus C_1^{(2)} \oplus B_1^{(1)} \oplus t_{6,1}. && (4.37)
\end{aligned}
$$

The complete attack is shown in Algorithm 4.2. Calculation of the time complexity is done in the same way as for the two-round attack with the help of the algorithm description:

$$
\frac{8 \cdot 2^8 + 6 \cdot 2^{16}}{6} \approx 2^{16}. \qquad (4.38)
$$

It is also clear from Algorithm 4.2 that no memory in addition to registers is needed to perform the attack. Like in the two-round case, the data complexity is 2.

Algorithm 4.2 Perform a known-plaintext attack on three-round SoDark-3 and print all candidate keys.

```
 1: procedure CrackThreeRounds($\mathcal{P}_1, \mathcal{C}_1, \mathcal{T}_1, \mathcal{P}_2, \mathcal{C}_2, \mathcal{T}_2$)
 2:     $\hat{B}_1^{(3)} \leftarrow s^{-1}(B_1^{(3)}) \oplus A_1^{(3)} \oplus C_1^{(3)} \oplus t_{1,1}$
 3:     $\hat{B}_2^{(3)} \leftarrow s^{-1}(B_2^{(3)}) \oplus A_2^{(3)} \oplus C_2^{(3)} \oplus t_{1,2}$
 4:     $\Delta B^{(2)} \leftarrow \hat{B}_1^{(3)} \oplus \hat{B}_2^{(3)}$
 5:     $\hat{A}_1^{(3)} \leftarrow s^{-1}(A_1^{(3)}) \oplus t_{7,1}$
 6:     $\hat{A}_2^{(3)} \leftarrow s^{-1}(A_2^{(3)}) \oplus t_{7,2}$
 7:     $\Delta A^{(2)} \leftarrow \hat{A}_1^{(3)} \oplus \hat{A}_2^{(3)} \oplus \Delta B^{(2)}$
 8:     $\hat{C}_1^{(3)} \leftarrow s^{-1}(C_1^{(3)}) \oplus t_{8,1}$
 9:     $\hat{C}_2^{(3)} \leftarrow s^{-1}(C_2^{(3)}) \oplus t_{8,2}$
10:     $\Delta C^{(2)} \leftarrow \hat{C}_1^{(3)} \oplus \hat{C}_2^{(3)} \oplus \Delta B^{(2)}$
11:     for all $k_2$ do                                              ▷ $2^8$ possible $k_2$
12:         $C_1^{(1)} \leftarrow s(C_1^{(0)} \oplus B_1^{(0)} \oplus k_2 \oplus t_{2,1})$
13:         $C_2^{(1)} \leftarrow s(C_2^{(0)} \oplus B_2^{(0)} \oplus k_2 \oplus t_{2,2})$
14:         $\Delta C^{(1)} \leftarrow C_1^{(1)} \oplus C_2^{(1)}$
15:         $B_1^{(2)} \leftarrow \hat{B}_1^{(3)} \oplus k_2$
16:         $B_2^{(2)} \leftarrow \hat{B}_2^{(3)} \oplus k_2$
17:         $\Delta B^{(1)} \leftarrow \Delta A^{(2)} \oplus \Delta C^{(2)} \oplus s^{-1}(B_1^{(2)}) \oplus s^{-1}(B_2^{(2)}) \oplus t_{6,1} \oplus t_{6,2}$
18:         for all $k_1$ do                                          ▷ $2^8$ possible $k_1$
19:             $C_1^{(2)} \leftarrow B_1^{(2)} \oplus \hat{C}_1^{(3)} \oplus k_1$
20:             $C_2^{(2)} \leftarrow B_2^{(2)} \oplus \hat{C}_2^{(3)} \oplus k_1$
21:             $\Delta \hat{C}^{(2)} \leftarrow s^{-1}(C_1^{(2)}) \oplus s^{-1}(C_2^{(2)}) \oplus t_{5,1} \oplus t_{5,2}$
22:             if $\Delta C^{(1)} = \Delta B^{(1)} \oplus \Delta \hat{C}^{(2)}$ then        ▷ True with probability $2^{-8}$
23:                 $A_1^{(1)} \leftarrow s(A_1^{(0)} \oplus B_1^{(0)} \oplus k_1 \oplus t_{1,1})$
24:                 $A_2^{(1)} \leftarrow s(A_2^{(0)} \oplus B_2^{(0)} \oplus k_1 \oplus t_{1,2})$
25:                 $\Delta A^{(1)} \leftarrow A_1^{(1)} \oplus A_2^{(1)}$
26:                 for all $k_7$ do                                  ▷ $2^8$ possible $k_7$
27:                     $A_1^{(2)} \leftarrow \hat{A}_1^{(3)} \oplus B_1^{(2)} \oplus k_7$
28:                     $A_2^{(2)} \leftarrow \hat{A}_2^{(3)} \oplus B_2^{(2)} \oplus k_7$
29:                     $\hat{A}_1^{(2)} \leftarrow s^{-1}(A_1^{(2)}) \oplus t_{4,1}$
30:                     $\hat{A}_2^{(2)} \leftarrow s^{-1}(A_2^{(2)}) \oplus t_{4,2}$
31:                     $\Delta \hat{A}^{(2)} \leftarrow \hat{A}_1^{(2)} \oplus \hat{A}_2^{(2)}$
```

```
32:                     if $\Delta A^{(1)} = \Delta B^{(1)} \oplus \Delta \hat{A}^{(2)}$ then          ▷ True with probability $2^{-8}$
33:                         for all $k_3$ do                                                        ▷ $2^8$ possible $k_3$
34:                             $k_{4,1} \leftarrow s(A_1^{(1)} \oplus C_1^{(1)} \oplus B_1^{(0)} \oplus k_3 \oplus t_{3,1}) \oplus \hat{A}_1^{(2)} \oplus A_1^{(1)}$
35:                             $k_{4,2} \leftarrow s(A_2^{(1)} \oplus C_2^{(1)} \oplus B_2^{(0)} \oplus k_3 \oplus t_{3,2}) \oplus \hat{A}_2^{(2)} \oplus A_2^{(1)}$
36:                             if $k_{4,1} = k_{4,2}$ then                                         ▷ True with probability $2^{-8}$
37:                                 $k_4 \leftarrow k_{4,1}$
38:                                 $k_5 \leftarrow s^{-1}(C_1^{(2)}) \oplus B_1^{(1)} \oplus C_1^{(1)} \oplus t_{5,1}$
39:                                 $k_6 \leftarrow s^{-1}(B_1^{(2)}) \oplus A_1^{(2)} \oplus C_1^{(2)} \oplus B_1^{(1)} \oplus t_{6,1}$
40:                                 PRINT($k_1 \parallel k_2 \parallel k_3 \parallel k_4 \parallel k_5 \parallel k_6 \parallel k_7$)
41:                             end if
42:                         end for
43:                     end if
44:                 end for
45:             end if
46:         end for
47:     end for
48: end procedure
```

4.5 Known-Plaintext Attack on Four-Round SoDark-3

Figure 4.3 shows the four-round attack. The basic principle of partial differential matching remains the same. This time the sieving is done using $\Delta A^{(2)}$.

The main loop of the attack iterates over all possible values of k_2 and k_3. In each loop, a list that associates the values of k_4 and $\Delta A^{(2)}$ with values of k_5, $\hat{A}^{(3)}$, $B^{(2)}$, and $\hat{C}^{(3)}$ is built from the ciphertexts using the following calculations and iterating over all values of k_4 and k_5:

$$B_1^{(3)} = s^{-1}\left(B_1^{(4)}\right) \oplus A_1^{(4)} \oplus C_1^{(4)} \oplus k_5 \oplus t_{4,1} \tag{4.39}$$

$$B_2^{(3)} = s^{-1}\left(B_2^{(4)}\right) \oplus A_2^{(4)} \oplus C_2^{(4)} \oplus k_5 \oplus t_{4,2} \tag{4.40}$$

$$A_1^{(3)} = s^{-1}\left(A_1^{(4)}\right) \oplus B_1^{(3)} \oplus k_3 \oplus t_{2,1} \tag{4.41}$$

$$A_2^{(3)} = s^{-1}\left(A_2^{(4)}\right) \oplus B_2^{(3)} \oplus k_3 \oplus t_{2,2} \tag{4.42}$$

$$C_1^{(3)} = s^{-1}\left(C_1^{(4)}\right) \oplus B_1^{(3)} \oplus k_4 \oplus t_{3,1} \tag{4.43}$$

$$C_2^{(3)} = s^{-1}\left(C_2^{(4)}\right) \oplus B_2^{(3)} \oplus k_4 \oplus t_{3,2} \tag{4.44}$$

$$B_1^{(2)} = s^{-1}\left(B_1^{(3)}\right) \oplus A_1^{(3)} \oplus C_1^{(3)} \oplus k_2 \oplus t_{1,1} \tag{4.45}$$

$$B_2^{(2)} = s^{-1}\left(B_2^{(3)}\right) \oplus A_2^{(3)} \oplus C_2^{(3)} \oplus k_2 \oplus t_{1,2} \tag{4.46}$$

$$\hat{A}_1^{(3)} = s^{-1}\left(A_1^{(3)}\right) \oplus B_1^{(2)} \oplus t_{7,1} \tag{4.47}$$

$$\hat{A}_2^{(3)} = s^{-1}\left(A_2^{(3)}\right) \oplus B_2^{(2)} \oplus t_{7,2} \tag{4.48}$$

$$\Delta A^{(2)} = \hat{A}_1^{(3)} \oplus \hat{A}_2^{(3)} \tag{4.49}$$

$$\hat{C}_1^{(3)} = s^{-1}\left(C_1^{(3)}\right) \oplus B_1^{(2)} \oplus t_{8,1} \tag{4.50}$$

$$\hat{C}_2^{(3)} = s^{-1}\left(C_2^{(3)}\right) \oplus B_2^{(2)} \oplus t_{8,2}. \tag{4.51}$$

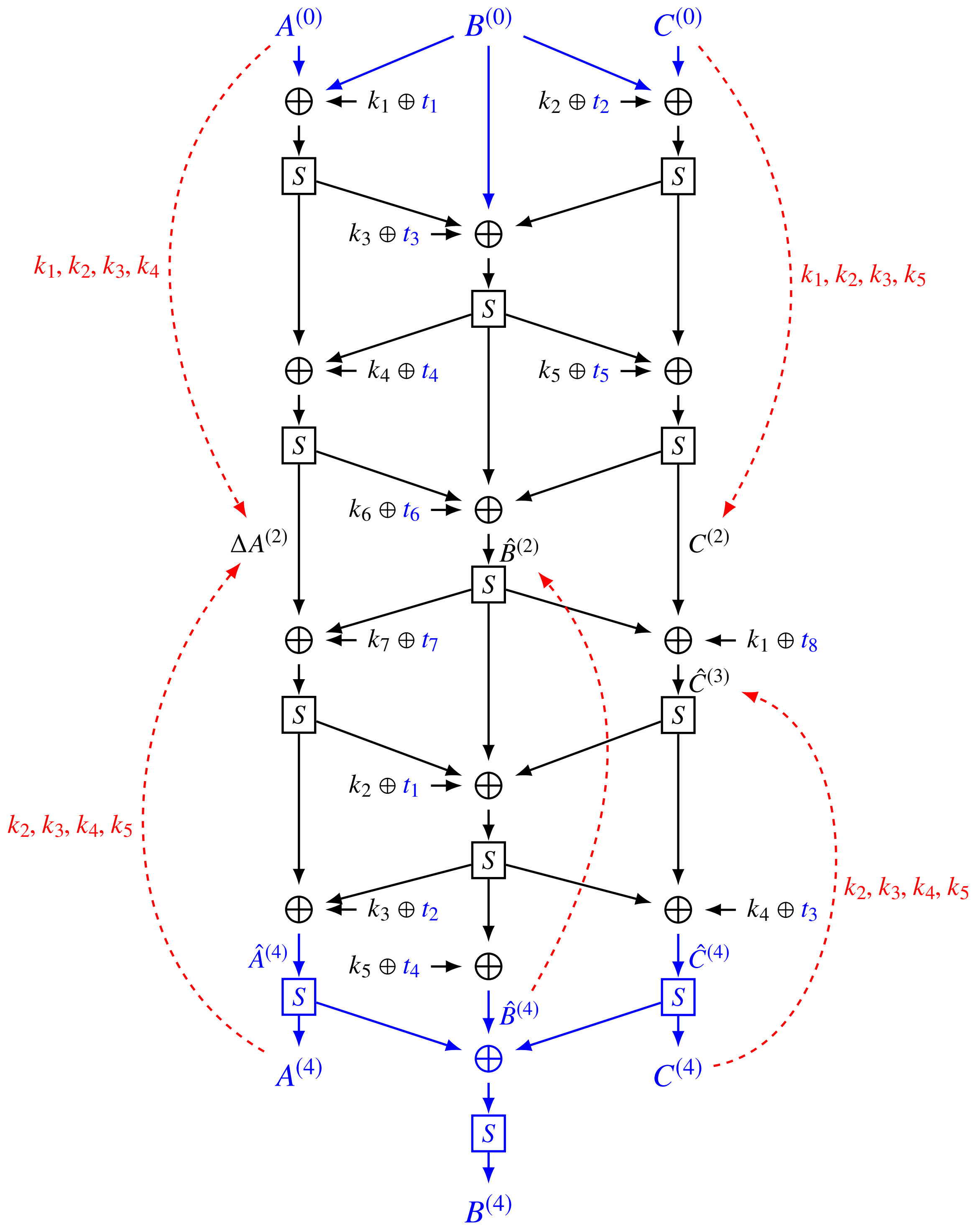

Figure 4.3. Attack on four-round SoDark-3 by matching on $\Delta A^{(2)}$. The parts of the cipher marked in blue are known or can be calculated without guessing any part of the key.

With the list built, the next step is to iterate over all possible values of k_1 and k_4 and calculate $\Delta A^{(2)}$ from the plaintexts:

$$\begin{aligned}
A_1^{(1)} &= s\left(A_1^{(0)} \oplus B_1^{(0)} \oplus k_1 \oplus t_{1,1}\right) && (4.52)\\
A_2^{(1)} &= s\left(A_2^{(0)} \oplus B_2^{(0)} \oplus k_1 \oplus t_{1,2}\right) && (4.53)\\
C_1^{(1)} &= s\left(C_1^{(0)} \oplus B_1^{(0)} \oplus k_2 \oplus t_{2,1}\right) && (4.54)\\
C_2^{(1)} &= s\left(C_2^{(0)} \oplus B_2^{(0)} \oplus k_2 \oplus t_{2,2}\right) && (4.55)\\
B_1^{(1)} &= s\left(A_1^{(1)} \oplus B_1^{(0)} \oplus C_1^{(1)} \oplus k_3 \oplus t_{3,1}\right) && (4.56)\\
B_2^{(1)} &= s\left(A_2^{(1)} \oplus B_2^{(0)} \oplus C_2^{(1)} \oplus k_3 \oplus t_{3,2}\right) && (4.57)\\
A_1^{(2)} &= s\left(A_1^{(1)} \oplus B_1^{(1)} \oplus k_4 \oplus t_{4,1}\right) && (4.58)\\
A_2^{(2)} &= s\left(A_2^{(1)} \oplus B_2^{(1)} \oplus k_4 \oplus t_{4,2}\right) && (4.59)\\
\Delta A^{(2)} &= A_1^{(2)} \oplus A_2^{(2)}. && (4.60)
\end{aligned}$$

For each value of $\Delta A^{(2)}$ calculated from the plaintext, the corresponding entries in the list calculated from the ciphertext are retrieved. Each entry will contain the implied value of k_5 and allow the calculation of k_1, k_6, and k_7:

$$\begin{aligned}
C_1^{(2)} &= s\left(B_1^{(1)} \oplus C_1^{(1)} \oplus k_5 \oplus t_{5,1}\right) && (4.61)\\
C_2^{(2)} &= s\left(B_2^{(1)} \oplus C_2^{(1)} \oplus k_5 \oplus t_{5,2}\right) && (4.62)\\
k_{1,1} &= C_1^{(2)} \oplus \hat{C}_1^{(3)} && (4.63)\\
k_{1,2} &= C_2^{(2)} \oplus \hat{C}_2^{(3)} && (4.64)\\
k_{6,1} &= B_1^{(1)} \oplus A_1^{(2)} \oplus C_1^{(2)} \oplus t_{6,1} \oplus s^{-1}\left(B^{(2)}\right) && (4.65)\\
k_{6,2} &= B_2^{(1)} \oplus A_2^{(2)} \oplus C_2^{(2)} \oplus t_{6,2} \oplus s^{-1}\left(B^{(2)}\right) && (4.66)\\
k_{7,1} &= A_1^{(2)} \oplus \hat{A}_1^{(3)} && (4.67)\\
k_{7,2} &= A_2^{(2)} \oplus \hat{A}_2^{(3)}. && (4.68)
\end{aligned}$$

Finally, good matches can be identified by checking that $k_1 = k_{1,1} = k_{1,2}$, $k_{6,1} = k_{6,2}$, and $k_{7,1} = k_{7,2}$. Algorithm 4.3 shows the attack process. That description is again used to calculate the time complexity of the attack which is

$$\frac{2 \cdot 2^8 + 4 \cdot 2^{24} + 8 \cdot 2^{32} + 4 \cdot 2.6 \cdot 2^{32}}{9} \approx 2^{32.9}. \tag{4.69}$$

The list used in the attack requires memory equivalent to about $2^{17.6}$ blocks. The data complexity remains 2.

Algorithm 4.3 Perform a known-plaintext attack on four-round SoDark-3 and print all candidate keys.

```
1: procedure CrackFourRounds($\mathcal{P}_1, \mathcal{C}_1, \mathcal{T}_1, \mathcal{P}_2, \mathcal{C}_2, \mathcal{T}_2$)
2:     $\hat{B}_1^{(4)} \leftarrow s^{-1}(B_1^{(4)}) \oplus A_1^{(4)} \oplus C_1^{(4)} \oplus t_{4,1}$
3:     $\hat{B}_2^{(4)} \leftarrow s^{-1}(B_2^{(4)}) \oplus A_2^{(4)} \oplus C_2^{(4)} \oplus t_{4,2}$
4:     $\hat{A}_1^{(4)} \leftarrow s^{-1}(A_1^{(4)}) \oplus t_{2,1}$
5:     $\hat{A}_2^{(4)} \leftarrow s^{-1}(A_2^{(4)}) \oplus t_{2,2}$
6:     $\hat{C}_1^{(4)} \leftarrow s^{-1}(C_1^{(4)}) \oplus t_{3,1}$
7:     $\hat{C}_2^{(4)} \leftarrow s^{-1}(C_2^{(4)}) \oplus t_{3,2}$
8:     for all $k_2$ do                                   ▷ $2^8$ possible $k_2$
9:         $C_1^{(1)} \leftarrow s(B_1 \oplus C_1 \oplus k_2 \oplus t_{2,1})$
10:        $C_2^{(1)} \leftarrow s(B_2 \oplus C_2 \oplus k_2 \oplus t_{2,2})$
11:        for all $k_3$ do                               ▷ $2^8$ possible $k_3$
12:            $L \leftarrow$ empty list                  ▷ Indexed by $k_4, \Delta A^{(2)}$
13:            for all $k_4, k_5$ do                      ▷ $2^{16}$ possible $k_4, k_5$
14:                $B_1^{(3)} \leftarrow \hat{B}_1^{(4)} \oplus k_5$
15:                $B_2^{(3)} \leftarrow \hat{B}_2^{(4)} \oplus k_5$
16:                $A_1^{(3)} \leftarrow \hat{A}_1^{(4)} \oplus B_1^{(3)} \oplus k_3$
17:                $A_2^{(3)} \leftarrow \hat{A}_2^{(4)} \oplus B_2^{(3)} \oplus k_3$
18:                $C_1^{(3)} \leftarrow \hat{C}_1^{(4)} \oplus B_1^{(3)} \oplus k_4$
19:                $C_2^{(3)} \leftarrow \hat{C}_2^{(4)} \oplus B_2^{(3)} \oplus k_4$
20:                $B_1^{(2)} \leftarrow s^{-1}(B_1^{(3)}) \oplus A_1^{(3)} \oplus C_1^{(3)} \oplus k_2 \oplus t_{1,1}$
21:                $B_2^{(2)} \leftarrow s^{-1}(B_2^{(3)}) \oplus A_2^{(3)} \oplus C_2^{(3)} \oplus k_2 \oplus t_{1,2}$
22:                $\hat{A}_1^{(3)} \leftarrow s^{-1}(A_1^{(3)}) \oplus B_1^{(2)} \oplus t_{7,1}$
23:                $\hat{A}_2^{(3)} \leftarrow s^{-1}(A_2^{(3)}) \oplus B_2^{(2)} \oplus t_{7,2}$
24:                $\hat{C}_1^{(3)} \leftarrow s^{-1}(C_1^{(3)}) \oplus B_1^{(2)} \oplus t_{8,1}$
25:                $\hat{C}_2^{(3)} \leftarrow s^{-1}(C_2^{(3)}) \oplus B_2^{(2)} \oplus t_{8,2}$
26:                $\Delta A^{(2)} \leftarrow \hat{A}_1^{(3)} \oplus \hat{A}_2^{(3)}$
27:                $L$.append($k_4, \Delta A^{(2)}, k_5, \hat{A}_1^{(3)}, \hat{A}_2^{(3)}, \hat{C}_1^{(3)}, \hat{C}_2^{(3)}, B_1^{(2)}, B_2^{(2)}$)
28:            end for
```

```
        for all $k_1$ do                                                    ▷ $2^8$ possible $k_1$
            $A_1^{(1)} \leftarrow s(A_1^{(0)} \oplus B_1^{(0)} \oplus k_1 \oplus t_{1,1})$
            $A_2^{(1)} \leftarrow s(A_2^{(0)} \oplus B_2^{(0)} \oplus k_1 \oplus t_{1,2})$
            $B_1^{(1)} \leftarrow s(B_1^{(0)} \oplus A_1^{(1)} \oplus C_1^{(1)} \oplus k_3 \oplus t_{3,1})$
            $B_2^{(1)} \leftarrow s(B_2^{(0)} \oplus A_2^{(1)} \oplus C_2^{(1)} \oplus k_3 \oplus t_{3,2})$
            for all $k_4$ do                                                ▷ $2^8$ possible $k_4$
                $A_1^{(2)} \leftarrow s(A_1^{(1)} \oplus B_1^{(1)} \oplus k_4 \oplus t_{4,1})$
                $A_2^{(2)} \leftarrow s(A_2^{(1)} \oplus B_2^{(1)} \oplus k_4 \oplus t_{4,2})$
                $\Delta A^{(2)} \leftarrow A_1^{(2)} \oplus A_2^{(2)}$
                for all $k_5, \hat{A}_1^{(3)}, \hat{A}_2^{(3)}, \hat{C}_1^{(3)}, \hat{C}_2^{(3)}, B_1^{(2)}, B_2^{(2)} \in L[k_4, \Delta A^{(2)}]$ do
                                                                    ▷ 2.6 iterations on average
                    $C_1^{(2)} \leftarrow s(B_1^{(1)} \oplus C_1^{(1)} \oplus k_5 \oplus t_{5,1})$
                    $C_2^{(2)} \leftarrow s(B_2^{(1)} \oplus C_2^{(1)} \oplus k_5 \oplus t_{5,2})$
                    $k_{1,1} \leftarrow C_1^{(2)} \oplus \hat{C}_1^{(3)}$
                    $k_{1,2} \leftarrow C_2^{(2)} \oplus \hat{C}_2^{(3)}$
                    $k_{6,1} \leftarrow s^{-1}(B_1^{(2)}) \oplus A_1^{(2)} \oplus C_1^{(2)} \oplus B_1^{(1)} \oplus t_{6,1}$
                    $k_{6,1} \leftarrow s^{-1}(B_2^{(2)}) \oplus A_2^{(2)} \oplus C_2^{(2)} \oplus B_2^{(1)} \oplus t_{6,2}$
                    $k_{7,1} \leftarrow A_1^{(2)} \oplus \hat{A}_1^{(3)}$
                    $k_{7,2} \leftarrow A_2^{(2)} \oplus \hat{A}_2^{(3)}$
                    if $k_1 = k_{1,1} = k_{1,2}$ and $k_{6,1} = k_{6,2}$ and $k_{7,1} = k_{7,2}$ then
                        $k_6 \leftarrow k_{6,1}$
                        $k_7 \leftarrow k_{7,1}$
                        PRINT($k_1 \parallel k_2 \parallel k_3 \parallel k_4 \parallel k_5 \parallel k_6 \parallel k_7$)
                    end if
                end for
            end for
        end for
    end for
  end for
end procedure
```

4.6 Known-Plaintext Attack on Five-Round SoDark-3

The attack on five-round SoDark-3 is structurally simpler than the previously described attacks and can be described entirely by treating the cipher as an iterated Even–Mansour construction. It is shown in Figure 4.4. Since the first two rounds use key bytes $k_1, k_2, k_3, k_4, k_5, k_6$ and the last two rounds use key bytes $k_1, k_3, k_4, k_5, k_6, k_7$, sieving can be done in the middle by comparing differentials, thus bypassing the third round key bytes. By looping over the common key bytes k_1, k_3, k_4, k_5, k_6 in an outer loop, the memory requirements are decreased significantly when compared to a standard MITM attack. The attack is equivalent to the three-subset MITM attack described in [37].

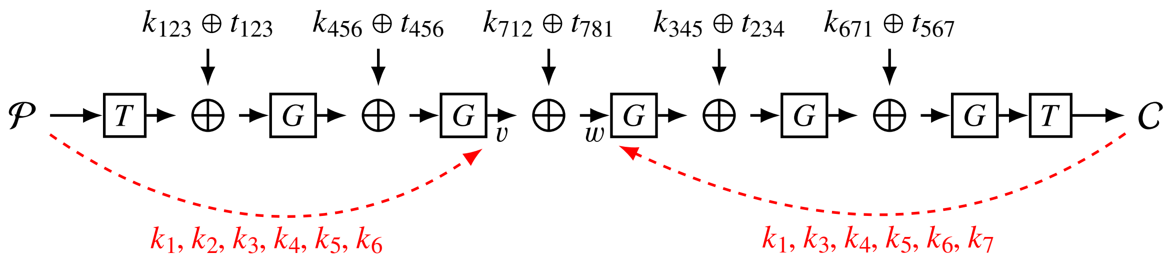

Figure 4.4. Attack on five-round SoDark-3.

Using the notation from Equation 3.24, the attack works by calculating

$$v_1 = g\big(g\big(T(\mathcal{P}_1) \oplus k_{123} \oplus t_{123,1}\big) \oplus k_{456} \oplus t_{456,1}\big) \oplus t_{781,1} \tag{4.70}$$

$$v_2 = g\big(g\big(T(\mathcal{P}_2) \oplus k_{123} \oplus t_{123,2}\big) \oplus k_{456} \oplus t_{456,2}\big) \oplus t_{781,2} \tag{4.71}$$

$$\Delta v = v_1 \oplus v_2 \tag{4.72}$$

for all possible values of $k_1, k_2, k_3, k_4, k_5, k_6$ and storing them in a list indexed by Δv. Then the same calculation is done for all possible values of $k_1, k_3, k_4, k_5, k_6, k_7$

$$w_1 = g^{-1}\Big(g^{-1}\Big(g^{-1}(T(C_1)) \oplus k_{671} \oplus t_{567,1}\Big) \oplus k_{345} \oplus t_{234,1}\Big) \tag{4.73}$$

$$w_2 = g^{-1}\Big(g^{-1}\Big(g^{-1}(T(C_2)) \oplus k_{671} \oplus t_{567,2}\Big) \oplus k_{345} \oplus t_{234,2}\Big) \tag{4.74}$$

$$\Delta w = w_1 \oplus w_2. \tag{4.75}$$

The value Δw is then used to look up the key bytes $k_1, k_2, k_3, k_4, k_5, k_6$ in the list. If the common key bytes k_2, k_3, k_4, k_5, k_6 match, the candidate key can be tested against more plaintext–ciphertext–tweak tuples.

Algorithm 4.4 implements the attack. The following time complexity is calculated from that algorithm:

$$\frac{12 \cdot 2^{40}\left(2^8 + 2^8\right)}{12} = 2^{49}. \tag{4.76}$$

The generated list uses 2^8 blocks of storage and the data complexity is still 2.

Algorithm 4.4 Perform a known-plaintext attack on five-round SoDark-3 and print all candidate keys.

```
1:  procedure CrackFiveRounds($\mathcal{P}_1, \mathcal{C}_1, \mathcal{T}_1, \mathcal{P}_2, \mathcal{C}_2, \mathcal{T}_2$)
2:      for all $k_1, k_3, k_4, k_5, k_6$ do                    ▷ $2^{40}$ possible $k_1, k_3, k_4, k_5, k_6$
3:          $L \leftarrow$ empty list
4:          for all $k_2$ do                                    ▷ $2^8$ possible $k_2$
5:              $v_1 \leftarrow g(g(T(\mathcal{P}_1) \oplus k_{123} \oplus t_{123,1}) \oplus k_{456} \oplus t_{456,1})$
6:              $v_2 \leftarrow g(g(T(\mathcal{P}_2) \oplus k_{123} \oplus t_{123,2}) \oplus k_{456} \oplus t_{456,2})$
7:              $\Delta v \leftarrow v_2 \oplus v_2$
8:              $L$.append($\Delta v, k_1$)
9:          end for
10:         for all $k_7$ do                                    ▷ $2^8$ possible $k_7$
11:             $w_1 \leftarrow g^{-1}(g^{-1}(g^{-1}(T(C_1)) \oplus k_{671} \oplus t_{567,1}) \oplus k_{345} \oplus t_{234,1})$
12:             $w_2 \leftarrow g^{-1}(g^{-1}(g^{-1}(T(C_2)) \oplus k_{671} \oplus t_{567,2}) \oplus k_{345} \oplus t_{234,2})$
13:             $\Delta w \leftarrow w_1 \oplus w_2$
14:             for all $k_1 \in L[\Delta w]$ do
15:                 Print($k_1 \parallel k_2 \parallel k_3 \parallel k_4 \parallel k_5 \parallel k_6 \parallel k_7$)
16:             end for
17:         end for
18:     end for
19: end procedure
```

4.7 Chosen-Tweak Attack on Six- and Seven-Round SoDark-3

Structural attacks of the types described previously, where the cipher is split in parts that use different subsets of the full seven-byte key, cannot be extended beyond five rounds. Nonetheless, for certain combinations of plaintext, ciphertext, tweak, and key, it is possible to predict part of the internal state of the cipher from the ciphertext alone.

For the six-round attack, consider two plaintext–ciphertext–tweak tuples where $\mathcal{P}_1 \neq \mathcal{P}_2$, $C_1 = C_2$ and all bytes in the tweak are identical except for $t_{5,1} \neq t_{5,2}$. The key schedule in Table 3.2 shows that this is possible if and only if $\Delta A^{(4)} = \Delta t_5 = t_{5,1} \oplus t_{5,2}$, $\Delta B^{(4)} = 0$, and $\Delta C^{(4)} = 0$. This known internal differential can be used to calculate $\Delta B^{(3)}$ and $\Delta C^{(3)}$ in the following way: First,

$$\Delta B^{(3)} = \Delta A^{(4)} \oplus s^{-1}\left(B_1^{(4)}\right) \oplus s^{-1}\left(B_2^{(4)}\right) \oplus \Delta C^{(4)} \oplus \Delta t_4. \tag{4.77}$$

Since $B_1^{(4)} = B_2^{(4)}$, $\Delta C^{(4)} = 0$, and $\Delta t_4 = 0$, this reduces Equation 4.77 to

$$\Delta B^{(3)} = \Delta A^{(4)} = \Delta t_5. \tag{4.78}$$

For the same reason,

$$\begin{aligned} \Delta C^{(3)} &= \Delta B^{(3)} \oplus s^{-1}\left(C_1^{(4)}\right) \oplus s^{-1}\left(C_2^{(4)}\right) \oplus \Delta t_3 \\ &= \Delta B^{(3)} = \Delta A^{(4)} = \Delta t_5. \end{aligned} \tag{4.79}$$

This knowledge allows sieving of possible $k_1, k_2, k_3, k_4, k_5, k_6$ by calculating $\Delta C^{(3)}$ from the plaintexts. The process is illustrated in Figure 4.5.

Unlike the previous attacks, which work on arbitrary message tuples, the attack on six rounds requires a specific output differential. The first step of the attack is therefore to find a plaintext–ciphertext–tweak tuple that satisfies it. Assuming that the cipher's randomization properties after four rounds are good,[1] all differentials after the fourth and subsequent rounds have probability 2^{-24}. The number of pairs of plaintext–ciphertext–tweak tuples n

[1]This is investigated in [5].

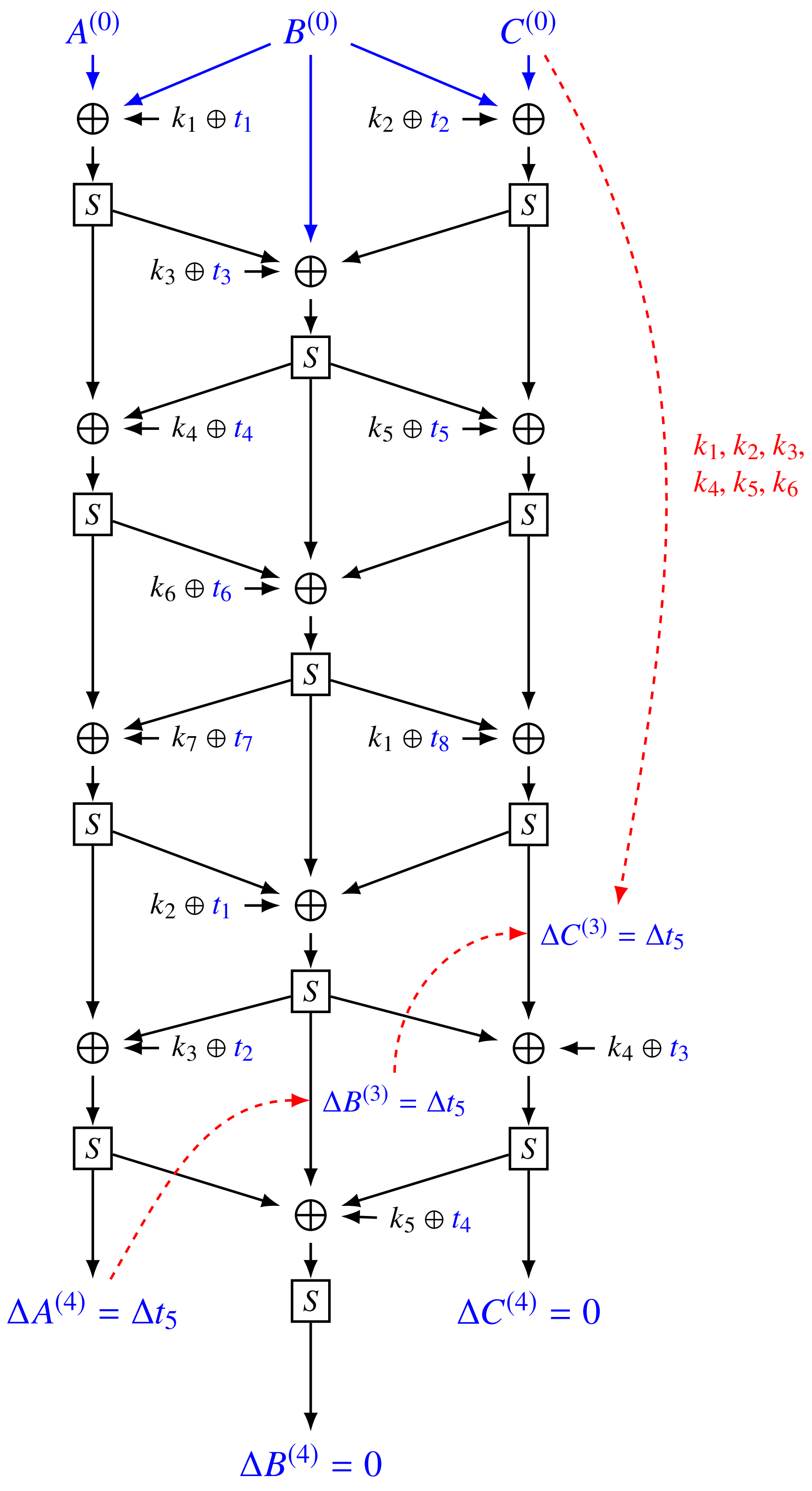

Figure 4.5. The first four rounds in the attacks on six- and seven-round SoDark-3.

required for one of them to have the required output differential with 50% probability is therefore

$$\left(1 - 2^{-24}\right)^n = \frac{1}{2} \implies n = \frac{\log \frac{1}{2}}{\log\left(2^{24} - 1\right) - \log\left(2^{24}\right)} \approx 11{,}629{,}080. \tag{4.80}$$

Unlike in a normal birthday attack, the required pairs of plaintext–ciphertext–tweak tuples must be formed so that each tuple in the pair has a different tweak. The most efficient way to achieve this in an oracle model is to generate plaintext–ciphertext–tweak tuples for two different tweaks with $t_{5,1} \neq t_{5,2}$ and all other tweak bytes identical. This way, with n generated tuples per tweak, n^2 tuple-pairs can be formed. Thus, only $\sqrt{11{,}629{,}080} \approx 3410 \approx 2^{11.7}$ tuples are required for each tweak in order to find the required output differential with 50% probability. This is, in effect, a version of the birthday paradox with two subsets.

Algorithm 4.5 performs the six-round attack. Since the filtering step can be done without any S-box operations, its time complexity can be neglected. The only source of complexity that remains is the calculation of $\Delta C^{(3)}$, which is

$$\frac{2 \cdot 2^8 + 2 \cdot 2^{16} + 2 \cdot 2^{24} + 2 \cdot 2^{32} + 2 \cdot 2^{40} + 4 \cdot 2^{48}}{15} \approx 2^{46.1}. \tag{4.81}$$

No memory in addition to registers is needed for the attack. Figure 4.6 shows the trade-off curve for the relationship between the number of available tuples and probability of success.

The attack is extended to seven rounds by the addition of an initial filtering step to find a pair with the correct fourth-round differential. For each generated pair of tuples, calculate

$$\Delta A^{(7)} = A_1^{(7)} \oplus A_2^{(7)} \tag{4.82}$$

$$\Delta C^{(7)} = C_1^{(7)} \oplus C_2^{(7)}. \tag{4.83}$$

If $\Delta A^{(7)} = 0$ and $\Delta C^{(7)} = 0$, continue by calculating

$$\hat{B}_1^{(7)} = s^{-1}\left(B_1^{(7)}\right) \oplus A_1^{(7)} \oplus C_1^{(7)} \oplus t_{5,1} \tag{4.84}$$

$$\hat{B}_2^{(7)} = s^{-1}\left(B_2^{(7)}\right) \oplus A_2^{(7)} \oplus C_2^{(7)} \oplus t_{5,2} \tag{4.85}$$

$$\Delta B^{(6)} = \hat{B}_1^{(7)} \oplus \hat{B}_2^{(7)}. \tag{4.86}$$

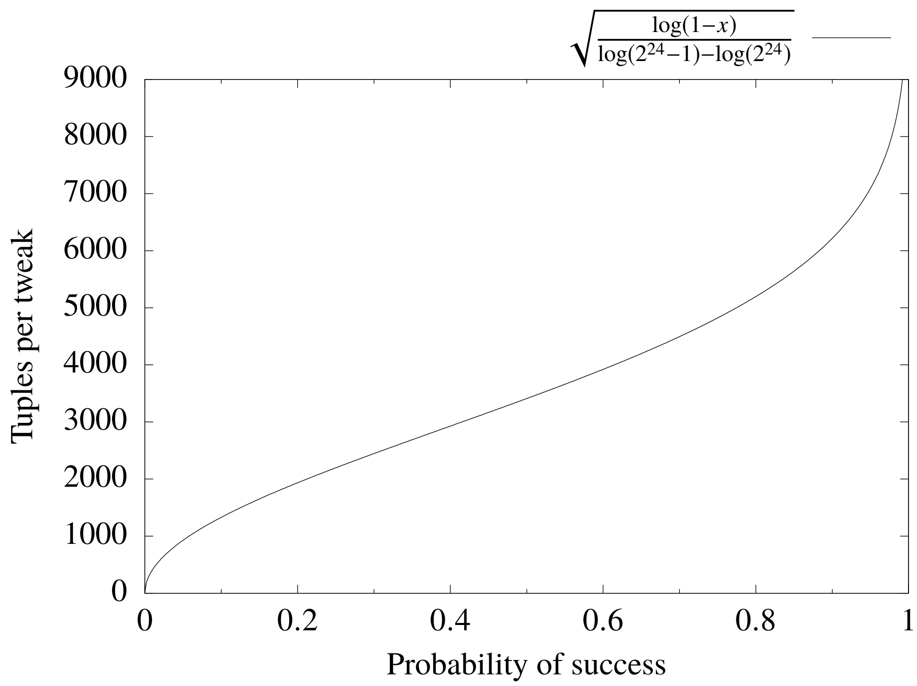

Figure 4.6. Trade-off curve between data and probability of success in the case with two sets of tweaks of same size.

If $\Delta B^{(6)} = 0$, the pair has the required fourth-round differential and the cipher can be attacked in the same way as the six-round version. Note that this filtering step does not involve guessing any key bits.

The addition of a filtering step increases the time complexity for an attack with 50% probability of success by a negligible amount:

$$\frac{2 \cdot 11{,}629{,}080 + 2 \cdot 2^8 + 2 \cdot 2^{16} + 2 \cdot 2^{24} + 2 \cdot 2^{32} + 2 \cdot 2^{40} + 4 \cdot 2^{48}}{18} \approx 2^{46.1}. \qquad (4.87)$$

Algorithm 4.5 Perform a chosen-tweak attack on six-round SoDark-3 and print all candidate keys.

```
Require: $C_1 = C_2, t_{1,1} = t_{1,2}, t_{2,1} = t_{2,2}, t_{3,1} = t_{3,2}, t_{4,1} = t_{4,2}, t_{5,1} \neq t_{5,2}, t_{6,1} = t_{6,2}, t_{7,1} = t_{7,2},$
    $t_{8,1} = t_{8,2}$
 1: procedure CrackSixRounds($\mathcal{P}_1, C_1, \mathcal{T}_1, \mathcal{P}_2, C_2, \mathcal{T}_2$)
 2:     $\Delta t_5 = t_{5,1} \oplus t_{5,2}$
 3:     for all $k_1$ do
 4:         $A_1^{(1)} \leftarrow s(A_1^{(0)} \oplus B_1^{(0)} \oplus k_1 \oplus t_{1,1})$
 5:         $A_2^{(1)} \leftarrow s(A_2^{(0)} \oplus B_2^{(0)} \oplus k_1 \oplus t_{1,2})$
 6:         for all $k_2$ do
 7:             $C_1^{(1)} \leftarrow s(C_1^{(0)} \oplus B_1^{(0)} \oplus k_2 \oplus t_{2,1})$
 8:             $C_2^{(1)} \leftarrow s(C_2^{(0)} \oplus B_2^{(0)} \oplus k_2 \oplus t_{2,2})$
 9:             for all $k_3$ do
10:                 $B_1^{(1)} \leftarrow s(A_1^{(1)} \oplus B_1^{(0)} \oplus C_1^{(1)} \oplus k_3 \oplus t_{3,1})$
11:                 $B_2^{(1)} \leftarrow s(A_2^{(1)} \oplus B_2^{(0)} \oplus C_2^{(1)} \oplus k_3 \oplus t_{3,2})$
12:                 for all $k_4$ do
13:                     $A_1^{(2)} \leftarrow s(A_1^{(1)} \oplus B_1^{(1)} \oplus k_4 \oplus t_{4,1})$
14:                     $A_2^{(2)} \leftarrow s(A_2^{(1)} \oplus B_2^{(1)} \oplus k_4 \oplus t_{4,2})$
15:                     for all $k_5$ do
16:                         $C_1^{(2)} \leftarrow s(C_1^{(1)} \oplus B_1^{(1)} \oplus k_5 \oplus t_{5,1})$
17:                         $C_2^{(2)} \leftarrow s(C_2^{(1)} \oplus B_2^{(1)} \oplus k_5 \oplus t_{5,2})$
18:                         for all $k_6$ do
19:                             $B_1^{(2)} \leftarrow s(A_1^{(2)} \oplus B_1^{(1)} \oplus C_1^{(2)} \oplus k_6 \oplus t_{6,1})$
20:                             $B_2^{(2)} \leftarrow s(A_2^{(2)} \oplus B_2^{(1)} \oplus C_2^{(2)} \oplus k_6 \oplus t_{6,2})$
21:                             $C_1^{(3)} \leftarrow s(C_1^{(2)} \oplus B_1^{(2)} \oplus k_1 \oplus t_{8,1})$
22:                             $C_2^{(3)} \leftarrow s(C_2^{(2)} \oplus B_2^{(2)} \oplus k_1 \oplus t_{8,2})$
23:                             $\Delta C^{(3)} \leftarrow C_1^{(3)} \oplus C_2^{(3)}$
24:                             if $\Delta C^{(3)} = \Delta t_5$ then
25:                                 Print($k_1 \parallel k_2 \parallel k_3 \parallel k_4 \parallel k_5 \parallel k_6$)
26:                             end if
27:                         end for
28:                     end for
29:                 end for
30:             end for
31:         end for
32:     end for
33: end procedure
```

4.8 Chosen-Tweak Attack on Eight-Round SoDark-3

The attack on six and seven rounds in the previous section can be extended to an attack on eight rounds, i.e., the full Lattice algorithm from the 2G ALE standard. Unlike the previous attack, the required output differential cannot be identified with certainty. It can, however, be identified with high probability. Figure 4.7 shows the last two rounds of the eight-round SoDark-3. The sought differential after the fourth round exists if and only if $\Delta A^{(7)} = \Delta C^{(7)} = 0$ and $\Delta \hat{B}^{(7)} = \Delta t_5$. In that case,

$$\Delta \hat{A}^{(8)} = \Delta A^{(7)} \oplus \Delta B^{(7)} = \Delta B^{(7)} \tag{4.88}$$

$$\Delta \hat{C}^{(8)} = \Delta C^{(7)} \oplus \Delta B^{(7)} = \Delta B^{(7)} \tag{4.89}$$

and therefore

$$\Delta \hat{A}^{(8)} = \Delta \hat{C}^{(8)} = \Delta B^{(7)}. \tag{4.90}$$

This differential just before the eighth round S-boxes therefore indicates a high probability that the seventh round differential is the required one. An attack that has 50% probability of success requires 11,629,080 plaintext–ciphertext–tweak tuple pairs, see Equation 4.80. The average number of candidate pairs remaining after the filtering step in that case is $2^{-16} \cdot 11{,}629{,}080 \approx 177.4 \approx 2^{7.5}$.

For plaintext–ciphertext–tweak tuples that satisfy Equation 4.90, the assumption is made that they have the correct fourth-round differential and the values of k_3 that cause

$$\begin{aligned} \Delta \hat{B}^{(7)} &= s^{-1}\left(s^{-1}\left(B_1^{(8)}\right) \oplus A_1^{(8)} \oplus C_1^{(8)} \oplus k_3 \oplus t_{8,1}\right) \oplus \\ &\quad s^{-1}\left(s^{-1}\left(B_2^{(8)}\right) \oplus A_2^{(8)} \oplus C_2^{(8)} \oplus k_3 \oplus t_{8,2}\right) \\ &= t_{5,1} \oplus t_{5,2} \end{aligned} \tag{4.91}$$

can be calculated. Candidate pairs remaining after the first filtering step will satisfy this relationship with probability $\frac{100}{256}$. In the 50% probability of success case, this will result in $177.4\frac{100}{256} \approx 69.3 \approx 2^{6.1}$ remaining candidate pairs.

For each remaining pair, the values of $k_1, k_2, k_3, k_4, k_5, k_6$ that give $\Delta C^{(3)} = \Delta t_5$ are searched for using the same method as in the previous six- and seven-round attack, with the exception

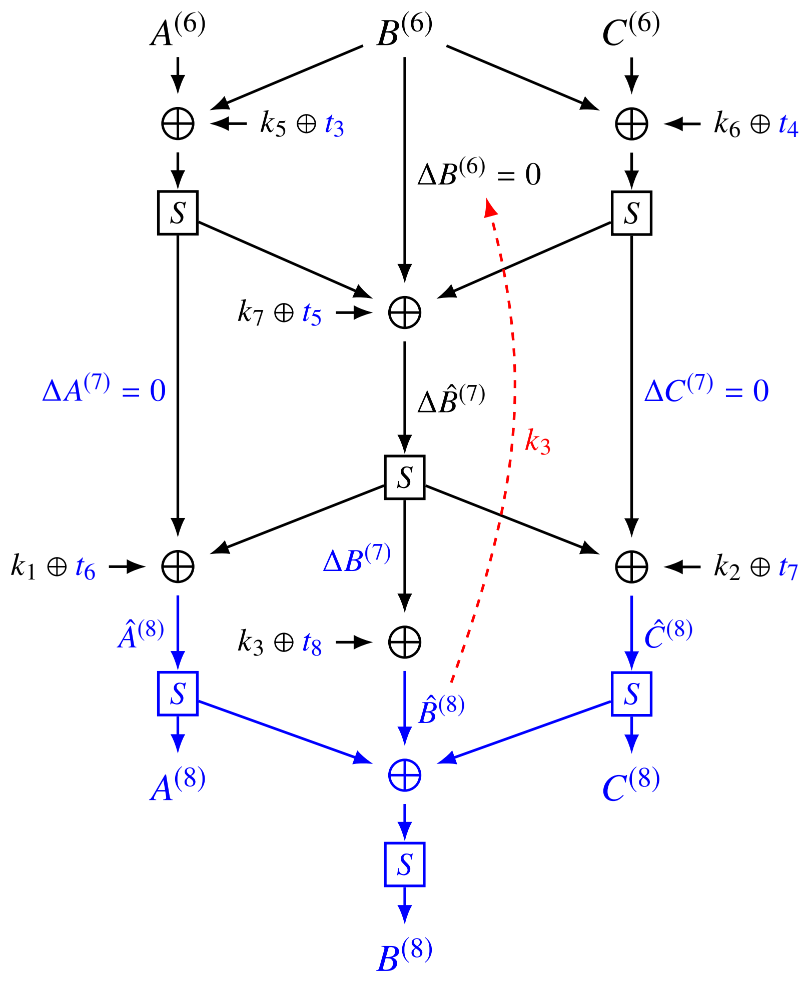

Figure 4.7. The last two rounds in the attack on eight-round SoDark-3.

that only the values of k_3 that satisfy Equation 4.91 for that pair are tried. We expect each candidate pair that survived filtering step two to have 2.6 candidate values for k_3 on average.

We can now calculate the total time complexity for the eight-round attack:

$$\begin{aligned}&\frac{1}{21} \cdot \Big(6 \cdot 11{,}629{,}080 + 4 \cdot 2^{7.5} \cdot 2^{8} + \\ &2^{6.1} \cdot \Big(2 \cdot 2^{8} + 2 \cdot 2^{16} + \frac{2.6}{2^{8}} \cdot \Big(2 \cdot 2^{24} + 2 \cdot 2^{32} + 2 \cdot 2^{40} + 4 \cdot 2^{48}\Big)\Big)\Big) \approx 2^{45.1}.\end{aligned} \tag{4.92}$$

The complexity of this attack is lower than the attacks on six and seven rounds presented in the previous section. This is because the differential after the next to last round—which is known with high probability—is used to deduce information about part of the key. Like in the six- and seven-round attacks, $2^{11.7}$ plaintext–ciphertext–tweak tuples are required to recover the key with 50% probability. The memory requirements also remain the same. No memory in addition to registers is required.

4.9 Experimental Verification

All attacks described in this chapter have been implemented in the C programming language and verified in practice. The implementations are publicly available [38].

CHAPTER 5: Logic Circuit Representations of the SoDark S-box

5.1 Introduction

For the attacks in the following chapters, an efficient logic circuit representation of the S-box is needed. Such a representation describes the relationship between the inputs and outputs of the S-box as a circuit of logic gates. A logic circuit implementation of an S-box considers each of the S-box output bits as a separate Boolean function of the same input variables. In the case of the SoDark S-box with eight inputs and eight outputs, this means eight Boolean functions of eight input variables. This is in contrast to representing the S-box as, for example, the algebraic normal form (ANF) of the Boolean functions it implements, or as a lookup table.

Since finding the optimum logic circuit for a given S-box is a NP-complete problem and is intractable even for very small S-boxes, heuristic methods must be used in all but very special cases. Although these heuristic methods are significantly faster than a brute force search, they are still quite slow and take a fair amount of time to perform, even on modern computers. In particular, for the logic circuit representations presented later that use 3-bit lookup tables (LUT), use of the NPS *Hamming* high-performance computing cluster was necessary.

In [19], Biham presents an algorithm for generating a logic circuit for the DES S-box. It breaks down the truth table of each Boolean function into 16 functions of two variables and then uses the remaining four "free" variables to choose between those 16 functions. Using this algorithm, Biham generates logic circuit representations of the DES S-box that require 100 gates on average.

It is important to note that, although a logic circuit with fewer gates is often faster, this is not always the case. Which circuit is faster in practice depends on the technology on which it is implemented. In the case of hardware implementations in ASICs and FPGAs, latency is normally of great concern. That means, the signal paths to different inputs of the same gate must have approximately the same delay. Too large a delay will necessitate a lower clock frequency and thus a lower speed.

Software implementations are typically limited by the available number of processor registers. This limits the number of gate outputs that are active in parallel. If the number of active outputs is higher than the number of available registers, memory must be used for storing the surplus output variables. This comes with a significant performance cost. Logic circuit representations used for generating algebraic systems as input to SAT solvers have similar problems. In that case, some Boolean gates result in CNF representations that are much easier for the SAT solvers to handle than others.

5.2 Kwan's Algorithm

In [39], Kwan presents an improvement to Biham's method from [19]. It works by successively adding new gates to a circuit through recursive search while trying all possible orderings of input and output bits. In this case, with eight input and eight output bits, it requires testing $8! \cdot 8!$ combinations.

The recursive algorithm described in [39] takes an existing partial gate circuit as input together with a target truth table, a "don't care" mask, and a list of input bits already used. It returns a gate in the circuit whose truth table is identical to the target, except for the bit positions where the don't care mask is zero. Initially, the gate circuit will only consist of the eight input bits.

Each invocation of the algorithm can be split up into five successively more complex steps [39]:

1. Check if there already is a gate in the logic circuit with the required output truth table. If so, return that gate.
2. Check if there is a gate with a truth table that is the logic NOT of the required output truth table. If so, add a NOT gate to the logic circuit and return it.
3. Try all combinations of two gates using AND, OR, XOR, NOT, and ANDNOT gates and check if the resulting output is equal to the target truth table. If so, add the gates and return the output gate.

4. Try all combinations of three gates using AND, OR, XOR, NOT, and ANDNOT gates. If one of the combinations results in the required output table, add the gates to the circuit and return the output gate.
5. Split the truth table on one of the unused input bits by setting the corresponding bits in the don't care mask to zero. Then call the algorithm twice recursively: once with a don't care mask corresponding to the input bit equal to one and once with a don't care mask corresponding to the input bit equal to zero. Combine the output from the two calls with a two gate multiplexer. Perform this once for each of the remaining unused input bits and with two different multiplexers. Return the combination of input bit and multiplexer that results in the logic circuit with the fewest gates.

The implementation details of Kwan's algorithm are somewhat complex and the reader is referred to [39] for a complete description.

The complexity of the algorithm increases for each of the five steps. The first and second steps have complexity $O(n)$, where n is the number of gates in the partial circuit. In step three, this increases to $O(n^2)$, since all possible combinations of two gates must be considered. For the same reason, the complexity of step four is $O(n^3)$. The most significant complexity is in step five: Due to the recursion in combination with the testing of all possible input bits, this results in a complexity of $O(b!)$, where b is the number of unused input bits. Even though the value of $b!$ is manageable in the case of the SoDark S-box, where $b \leq 8$, the big O notation hides the high complexity of each individual recursive call, which can include a complexity of $O(n^3)$ in addition to the $O(b!)$ term.

Finding the most efficient logic circuit for all eight output functions requires testing all 8! orders of building those eight output functions. The result is a total complexity of Kwan's algorithm of $O(b! \cdot b!)$, where b is the number of S-box input and output bits.

5.3 Improvements to Kwan's Algorithm

An anonymous software project for building three-bit LUT circuit representations of S-boxes is available as a GitHub repository [40]. It contains several improvements to Kwan's algorithm.

Apart from the generation of LUT-based logic circuits, the two major improvements to Kwan's algorithm introduced in [40] are circuit randomization and a fast feasibility checking algorithm.

The algorithm described by Kwan is deterministic and will always produce the same output given the same input. Due to the heuristic nature of the algorithm, there is no guarantee that this is the optimal result. By introducing randomization of the search order when searching for combinations of gates in steps one through four in the previous section, we can find equivalent—and possibly better—gate circuits simply by running the algorithm several times.

The fast feasibility checking algorithm described in Algorithm 5.1 significantly improves the speed of Kwan's algorithm by short-circuiting parts of step four in the previous section. It does this by performing a constant-time feasibility check for each combination of three gates before testing a large number of possible ways to combine them. The feasibility check itself is due to an interesting observation: Three gates with arbitrary truth tables can be combined to form an arbitrary target truth table if and only if the target truth table can be expressed as a product-of-sums expansion of the three input truth tables [41]. The feasibility checking algorithm can be extended and applied to an arbitrary number of input gates in a straightforward way.

When generating LUT-based circuits, additional steps are added between steps four and five in Kwan's algorithm. These steps search for combinations of three, five, and seven gates together with one, two, and three LUTs, respectively, to create the desired output truth table. Considering the large complexities involved in searching through all possible combinations of five and seven gates in the partial circuit, this would not be possible without the speed increase provided by the fast feasibility checking algorithm, especially considering that there are 256 possible functions for each LUT.

Algorithm 5.1 Check if a target truth table can be produced by combining three input truth tables using any combination of gates. Adapted from [40].

```
 1: procedure Check3Possible(target, mask, table1, table2, table3)
 2:     match ← 0
 3:     $t_1$ ← NOT table1
 4:     $i \leftarrow 0$
 5:     for $i < 2$ do
 6:         $t_2$ ← NOT table2
 7:         $k \leftarrow 0$
 8:         for $k < 2$ do
 9:             $t_3$ ← NOT table3
10:             $m \leftarrow 0$
11:             for $m < 2$ do
12:                 $r \leftarrow t_1$ AND $t_2$ AND $t_3$
13:                 if (target AND $r$ AND mask) = ($r$ AND mask) then
14:                     match ← match OR $r$
15:                 else if (target AND $r$ AND mask) $\neq 0$ then
16:                     return false
17:                 end if
18:                 $t_3$ ← NOT $t_3$
19:                 $m \leftarrow m + 1$
20:             end for
21:             $t_2$ ← NOT $t_2$
22:             $k \leftarrow k + 1$
23:         end for
24:         $t_1$ ← NOT $t_1$
25:         $i \leftarrow i + 1$
26:     end for
27:     if (target AND mask) = (match AND mask) then
28:         return true
29:     else
30:         return false
31:     end if
32: end procedure
```

5.4 Software Implementation

For this research, Kwan's algorithm from [39], along with the optimizations and modifications from [40], was implemented in the C programming language [42]. The resulting program can find logic circuit representations of the SoDark S-box that are suitable for various types of implementations on different platforms. This includes representations that use only the standard AND, OR, NOT, and XOR gates as well as an option that also allows for ANDNOT gates. Circuits can be built for a single output bit each, or for any combination of output bits.

In addition to using the number of gates as a metric when building the circuits, a metric that promotes circuits with efficient CNF representations is also available. The latter is intended for generating S-box circuit representations that have high performance when used with SAT solvers. It uses the number of three-variable minterms in the CNF representation of the logic circuit as a measure of the circuit's SAT performance.

Circuits of 3-bit LUTs can also be generated. This allows fast bitslicing implementations on Nvidia platforms that implement the `lop3.b32` Parallel Thread Execution (PTX) instruction, as described in Chapter 7. The logic circuits generated by the program can be output as C or CUDA source code as well as in the Graphwiz [43] DOT format for visualization.

5.5 Generated Circuits

The program described in the previous section was used to generate circuits for the S-box that are suitable for implementations on general purpose computers, CUDA GPUs, and for conversion to CNF for use with SAT solvers. Despite the optimizations made, and the use of 1024 processor cores on the *Hamming* high-performance computer cluster, creating a combined logic circuit for all eight Boolean functions using LUTs proved to be too large a problem. Instead, eight separate circuits were created. Figures 5.1 and 5.2 show examples of the generated circuits.

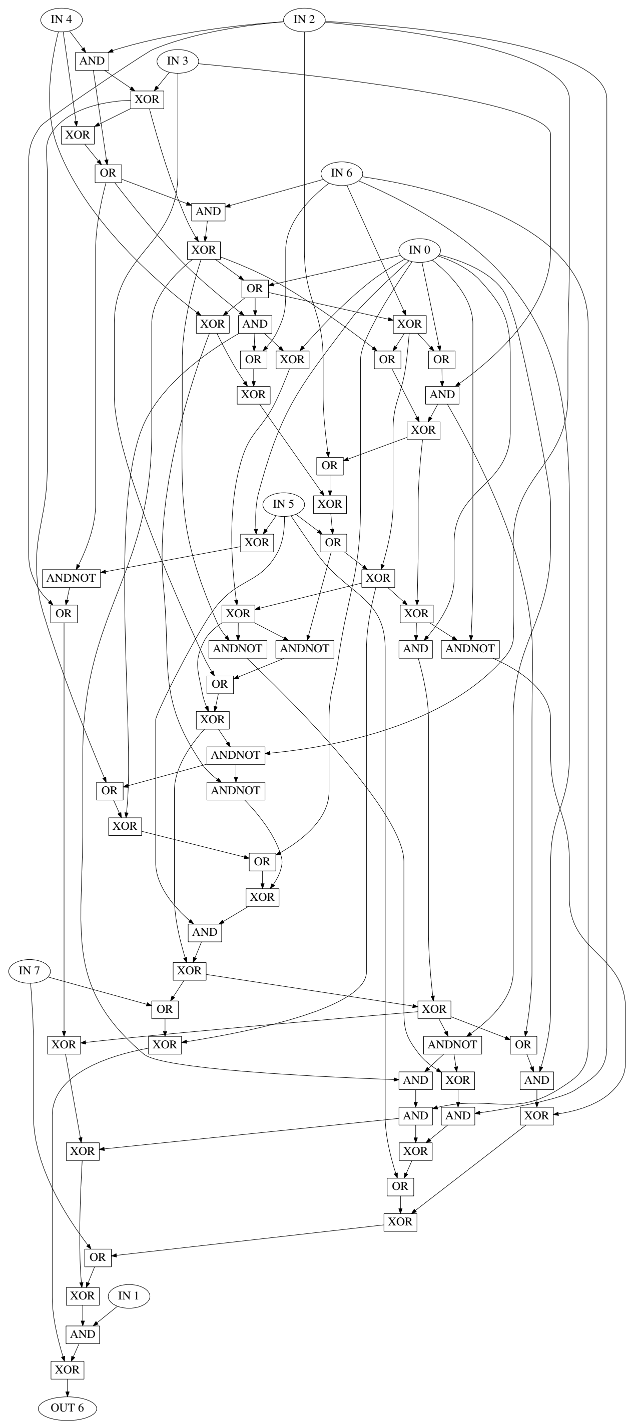

Figure 5.1. Logic circuit representation, with 60 gates, of the Boolean function for output bit 6 of the SoDark S-box.

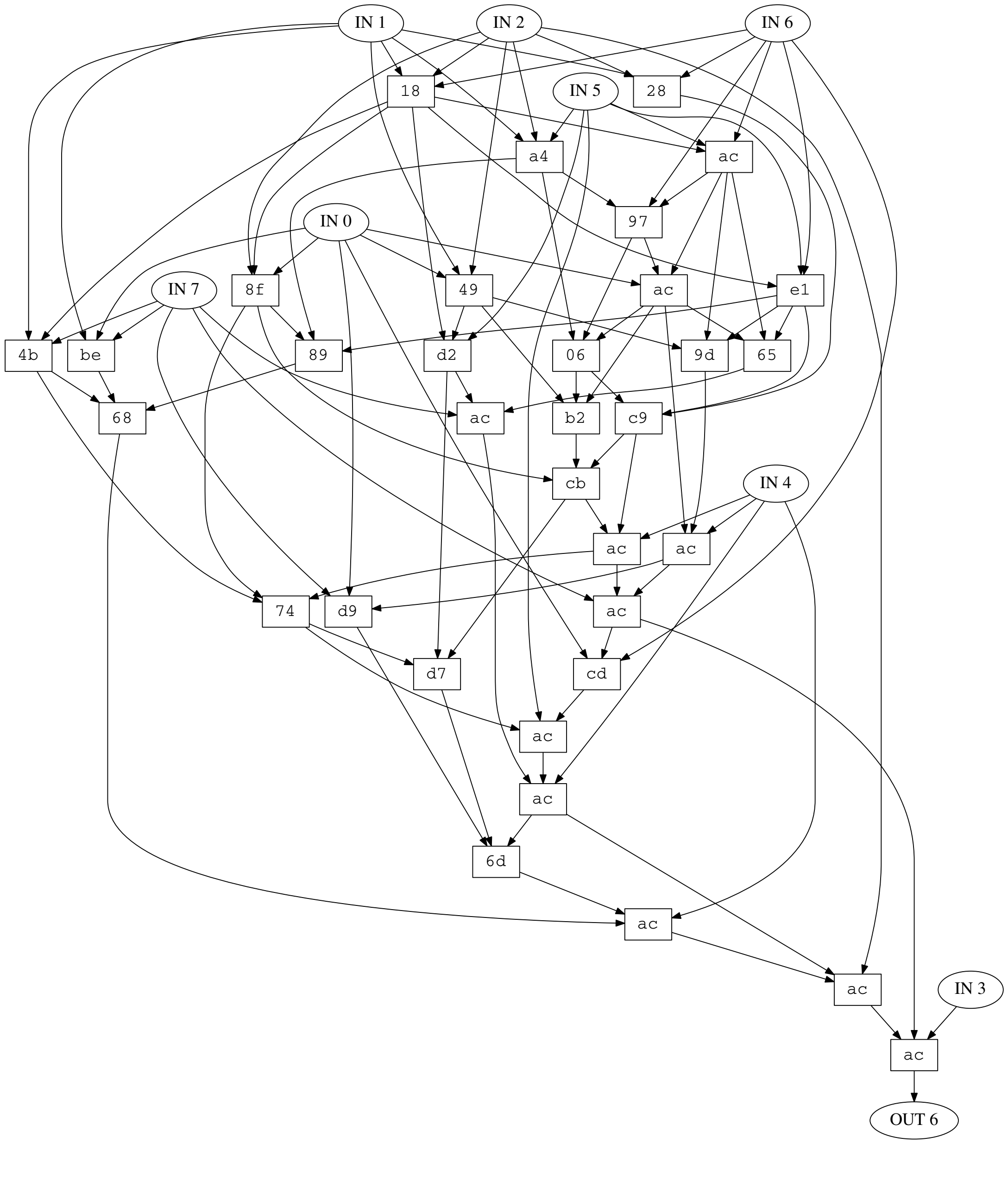

Figure 5.2. LUT circuit representation, with 34 LUTs, of the Boolean function for output bit 6 of the SoDark S-box. The number in each box is the hexadecimal representation of the LUT truth table.

CHAPTER 6: SAT-Based Attacks

The SAT problem is a fundamental problem of computer science. The description of the problem is simple: Given a Boolean formula, is there an assignment to its variables for which the formula evaluates to true? If such an assignment exists, the formula is said to be satisfiable. SAT problems are normally stated in CNF form. If the problem can be stated in a form where none of the minterms in its CNF expression contains more than two variables, it is said to be a 2-CNF SAT problem. Solutions for 2-CNF SAT problems can be found in polynomial time.

The definition of the 3-CNF SAT problem is analogous to the 2-CNF SAT definition and it has been proven that all SAT problems of higher order are reducible to an equivalent 3-CNF SAT problem. Furthermore, the 3-CNF SAT problem is proven NP-complete and is among the most studied NP problems [44]. The worst-case performance of the 3-CNF SAT is the same as for other MQ problems: $O(2^{\alpha n})$, $0 < \alpha \leq 1$. With modern SAT-solvers, $\alpha \geq 0.386$ for satisfiable 3-CNF SAT problems. Problems encountered in practice can often be solved even faster than this [15].

SAT solvers are computer programs specifically developed for solving SAT problems. Modern SAT solvers can solve hard problems involving thousands of variables occurring in a wide range of applications. In contrast, naïve brute force methods can handle only a few tens of variables. The construction of SAT solving algorithms is still an active research problem in academia and many different heuristics are used. For that reason, this research's focus has been on creating efficient CNF representations while treating the SAT solvers as black boxes.

The problem of recovering the key from a cipher can be converted into a SAT problem by expressing the entire cipher in CNF. The logic circuit representations of the SoDark S-box created in Chapter 5 can be converted into CNF by using the Tseytin transform [45] whereby the gates in the circuit are converted to equivalent CNF representations. Table 6.1 shows CNF representations of the gates used in Chapter 5.

Table 6.1. Tseytin transformations for some logic gates. Adapted from [45].

Logic gate	Operation	Conjunctive normal form
NOT	$C = \overline{A}$	$(\overline{A} \vee \overline{C}) \wedge (A \vee C)$
AND	$C = A \cdot B$	$(\overline{A} \vee \overline{B} \vee C) \wedge (A \vee \overline{C}) \wedge (B \vee \overline{C})$
OR	$C = A + B$	$(A \vee B \vee \overline{C}) \wedge (\overline{A} \vee C) \wedge (\overline{B} \vee C)$
XOR	$C = A \oplus B$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee B \vee \overline{C}) \wedge (A \vee \overline{B} \vee C) \wedge (\overline{A} \vee B \vee C)$
ANDNOT	$C = A \cdot \overline{B}$	$(\overline{A} \vee B \vee C) \wedge (A \vee \overline{C}) \wedge (\overline{B} \vee \overline{C})$

A C program that constructs a problem for input to a SAT solver was created [38]. It takes three plaintext–ciphertext–tweak tuples as input and converts them to their respective implied CNF representations in the DIMACS format commonly used by SAT solvers. Except for the S-boxes, the cipher consists entirely of XOR operations. This makes the conversion process fairly simple. It consists of converting Equations 3.1, 3.2, and 3.3 for each round into CNF using the logic circuit representation from Chapter 5 and the Tseytin transformations of the operations from Table 6.1. The 56 variables representing the key bits are shared among the parallel cipher representations. The 64 tweak bits can be completely removed from the CNF representation by observing that the XOR addition of a known bit is equivalent to the NOT operation if the bit is one and to doing nothing if the bit is zero.

If the plaintext–ciphertext–tweak tuples are correct, the constructed SAT problem will be satisfiable. Due to the small block size, three tuples are needed to imply a single key in the case of SoDark-3.

Table 6.2 shows statistics of the CNF representations for various numbers of rounds. The representations of the test vectors from [5] were tested with three different SAT solvers: CryptoMiniSat [28], Plingeling, and Treengeling [29]. All three are state-of-the-art parallel solvers that have performed well in the International SAT Competitions [46]. Plingeling and Treengeling are part of the Lingeling family of SAT solvers, while CryptoMiniSat is a fork of MiniSat [47] optimized for solving cryptological problems.

Table 6.2. CNF representation statistics.

Rounds	Clauses	Variables
2	3864	12438
3	7479	24252
4	11094	36066
5	14709	47880

PLINGELING and TREENGELING were successful in solving the SAT problems and recovering the key for up to four rounds while CRYPTOMINISAT only managed to solve the two- and three-round SAT problems. For five-round problems, none of the solvers could find solutions even after more than two weeks of search. Solution times for each of the solvers are plotted in Figure 6.1.

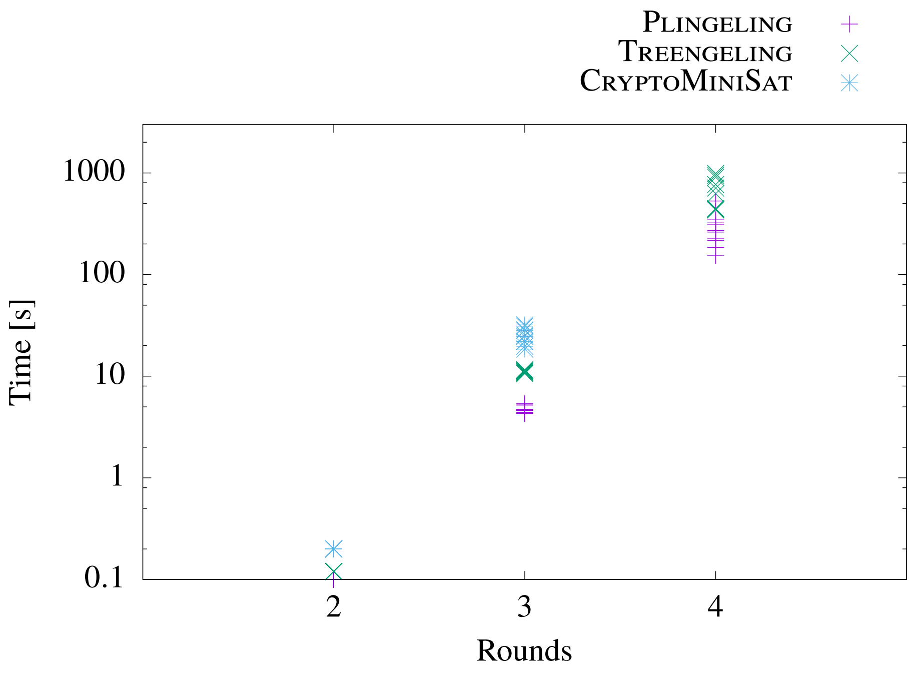

Figure 6.1. Performance of SAT solver attacks.

CHAPTER 7:
Brute Force Attacks

7.1 Introduction

As said in Chapter 2, all ciphers can be broken by brute force. For that reason, the size of the key space must be large enough to prevent exhaustive key search. With their small key lengths of 56 bits, all SoDark variants can be assumed to be vulnerable to brute force attacks in practice, see Table 2.3. For that reason, and to measure the actual upper bound of security for the algorithm, a brute force attack was mounted.

An efficient brute force attack necessitates a fast implementation of the cipher. Section 2.4.1 discusses some different approaches to fast exhaustive key search. From the investigations so far, nothing has been uncovered that would prevent ASIC attacks from being successful with speeds in the same order as the EFF's ASIC-based computer on DES. Time and resource limitations prohibit such an attempt in this case and restrict attempts to commonly available computer hardware.

7.2 The CUDA Framework

The Nvidia CUDA parallel computing framework was chosen for the brute force implementation. It is a GPGPU framework primarily designed for use with Nvidia's various GPU products and provides a C-like programming language for writing programs that run on them. A feature of recent generations of Nvidia GPUs that make them particularly suitable for brute force key search is the `lop3.b32` PTX instruction. PTX is the intermediate assembly-like language used by the CUDA framework and its `lop3.b32` instruction performs a bitwise 3-bit table lookup [48]. This single-instruction bitwise lookup enables the creation of bitslicing implementations that are faster compared to implementations that use only standard bitwise logic instructions.

The execution of GPGPU programs differs significantly from the execution of programs on normal CPUs. GPUs can have thousands of cores and are therefore able to execute thousands of concurrent threads. Unlike CPU cores, the GPU cores execute in lockstep.

While this is one of the reasons behind the speeds provided by GPGPU computing, it also causes severe performance penalties for branching instructions. Fast GPGPU programs therefore limit, or preferably eliminate, branch instructions. Performing operations on the processor and GPU in parallel with copying data between computer and GPU memory also improves performance by reducing latency [49].

7.3 Brute Force Bitslicing Implementation

A CUDA bitslicing implementation of SoDark was developed for this thesis [38] using all methods described in the previous section to achieve close to optimal performance. It takes two or three plaintext–ciphertext–tweak tuples as input and outputs all matching keys. It supports using several CUDA devices and launches three parallel CPU threads per GPU device in order to minimize latency.

The key space is divided into 2^{24} subsets of 2^{32} keys each. All keys in each set of 2^{32} keys share the same three most significant key bytes. This means that the first round, which uses only those key bytes, can be calculated once for all keys in the set. In the case of the Lattice eight-round version, the same applies to the last round, see Table 3.2. This reduces the number of rounds that the bitslicing part of the implementation has to perform from eight to six. Importantly, only five rounds of S-box operations have to be performed.

With the guessed states after the first and before the last round having been computed on the CPU, the rest of the key bytes are tested on the GPU using a carefully optimized branch-free bitslicing CUDA implementation of rounds two through six. Since the platform register size is 32 bits, each kernel iteration tests 32 keys in parallel. Instead of executing branch instructions on the GPU to test for expected output, the comparison is done using bitwise logic instructions and the result copied to GPU memory. After each kernel is finished executing for a certain subset of 2^{32} keys, the results are copied from GPU to computer memory while another kernel executes for the next subset of keys.

After copying the results to main memory, the CPU checks for keys that matched the first plaintext–ciphertext–tweak tuple. Matches are verified against the second and third tuples using a CPU SoDark implementation. Keys that satisfy all three tuples are output as candidates.

7.4 Attack in Practice

An exhaustive key search for all possible keys satisfying the first two plaintext–ciphertext–tweak tuples from the test vectors in [5] was performed using the implementation described in the previous section. The computer used had three Nvidia GeForce GTX 1070 GPUs. The entire key space took 14 days to search through. All keys matching the two tuples are presented in Table 7.1. This effectively proves that an exhaustive key search has been successfully performed.

7.5 Ciphertext-Only Attack

The known-plaintext brute force attacks on the 2G and 3G ALE linking protection ciphers can be extended to ciphertext-only attacks. This is made possible by the stereotypical nature of ALE linking operations and PDU format. In many cases, parts of the plaintexts can be accurately guessed only by observing encrypted message traffic. For a normal 2G ALE link establishment call as described in Section 2.3, the three bit preamble of each plaintext will be known. Additionally, it is known that the addresses in the first two PDUs will be identical. In total, this equals about 30 bits of known information that can be used in a brute force attack. More collected ciphertext–tweak pairs will be needed than for the corresponding known-plaintext attack, in order to reduce the set of candidate keys to a manageable number.

Ciphertext-only attacks become easier with 3G ALE PDUs. This is due to the fact that only 24 bits of the 26-bit PDUs are encrypted. The two unencrypted bits, together with observations on the encrypted traffic, give information about the structure of the plaintext allowing bits to be guessed. Furthermore, the inclusion of a CRC value in the plaintext allows for easy validity checking of plaintexts during the attack.

Table 7.1. All 218 keys that satisfy the first two LATTICE test vectors ([54E0CD, C0D705, 543BD88000017550] and [54E0CD, 708434, 543BD88040017550]), obtained through exhaustive search.

00b07038aec462	3069d03b5e6bd1	636ba0f5daa068	95d6bf32317e41	ceac62ee56d694
0140e8f5915b5e	309e68761e9dd7	63775f3100a06b	97ac52e6bfa80d	cf013ed902f61f
015975ce6b7331	328e55ef58b38f	639575c7a443b7	97c24df174a0ae	d28b7ee995bc99
01fec4c977db1a	330a6144e0f90a	6509bbd7024c00	9821193e93d9a6	d46cd2d8f55eed
054e2802f89744	34e7ee30476031	66eb40e2733b91	985ed4ca1472dc	d513cc98aa5302
057ce95bbf0ba0	36ed58e321f6f5	674124a9bbc4ad	987e053a3b0878	d6a1f169f258a2
0673dcd87f4713	37e02d9cdb17c4	679f47c6d17629	9a5fea12c494ce	d6a54180d05bf9
06f74260f1fb20	381cff4ee924a9	6970f4b6091186	9af042a22bcc6d	d6db6c4f39c164
0736fc708afcfb	3bf99f5df6d293	6b5c659d868047	9cd6cb701c6e5d	d6ff7b2ecac95e
093f191fd0c370	3c97254421af02	6b751774cdcae0	9dee63f1597bc0	d7b5481ec6da24
09b97c5da4480e	3cd3f0019caa90	6d37f4f1cccb67	9e09ec9644f69f	d7fc96b6f571c5
0a150dcde7a946	41d252725166d7	6d5fd40e35fa7b	9e327157d939c4	d8ce546516e0ac
0b21be27bb3af5	4260a04cf6cd2e	6ea860546f6ab3	9f202c12809cc5	d9e49aa6dfce69
0b34fbf6ae7656	42d7abbb7dc6c3	6ff389ec21f5a9	9fdd172904cb6b	daabe8dffe568b
0baa2008befd33	43130e6780ebdf	7080882885bbe1	a256caef0a97a9	dbf0e44ac9666a
0d3b69201c3a6a	43231efabca62b	7112618dc6db8a	a38b32d6025dab	dc474d99989616
0d8d8a7caaf04d	43fdf165c2ba8a	71c8bb8ffc20d2	a50411d787d7d2	dd21ff4906eff0
0df3956499fd9d	44463166ed6f0d	724c0d46b55d70	a563907ef6b053	e1edb57d6bb1ce
0f1ec077dd9bd6	44cbf7563ea50f	7310a1f76669c8	a6b4fe3d432e61	e22d507a6da9e3
0f83ac243b6f48	44cdc9cd84837b	73df341b32f237	a903a2be1e5ff3	e23a9e59b81e17
0f84c9ecce9b52	457bd494d7f982	73efe6c86b7bdf	aaad65d424e4b7	e28e10822e4d7a
11d421ea82165d	4763aca9ef0eaa	73fdcb5a227163	ae11a11cd353d7	e59eba9300481f
12e272992b13ef	49eb4a3a0d8efe	77455402f69746	ae1391092ec654	e68e20381d0286
1480b2fd91ab16	4a4e1a99da0bb6	79015483cfe3f3	ae5b64ae8f9272	e7204957893dc7
14935eba42dd80	4d31490675fa0d	7b55c8bfd6b858	b00bc84f7637a3	e856d002e1b97c
160fc808e98a89	4e42de4df4e043	7bfe33a5ba521e	b0be6612d34c44	e946715362f1d7
172bf130e2b1aa	4f2f82502c6ef2	7d3599e156f52d	b10f1184b0ba4a	eaa2198c885feb
18e2d0ce88e921	5069e3bab80432	7e4dc45bccf57d	b11b1afa058f2c	eab311d7b5613c
1a2eac9a1915f7	51a82254021c3f	7e55559c8ba98f	b3d46d242c10fb	ee08b15b35e8cb
1a734efc2cd7c4	52eafaf530e9b0	7ed4507fd839c8	b6ab25108eec73	f0bc6d18038540
1c80b0e0468236	53e62ea0282fe9	82413525e542f4	bc5fb2c66ee64f	f15bfd12e901b5
1cc269c99f364a	545d15797d2949	853d879087ff0b	bc61e02a245b12	f488ff76fd81cd
1eac88355d276c	55f76c00a23f7d	8566527368ebc7	bc7d8d40512387	f51665c1eacc5e
1f5b56daf7c390	56f1093816d005	86be2141366558	bdedf33da8cce7	f6c5ac930a1bad
1fe48727721d6e	5773398aaf380a	876dcbf8367082	c2284a1ce7be2f	f7aef7f9d5e1e2
21aff020b0970e	590731e28cd161	894b2ec0f7c881	c24f3751b51f64	f971ffa5233a36
22319c966def7b	5b394bd050a895	8d060177eaa07d	c32497bb9f05b4	f9a612a8d08ba9
26375adab9bb06	5c21f59d23ec58	8e3ab8da6862ee	c4e94b7424f87f	fa28f3c61cb0a9
297e454a67b337	5c54214862f1b8	8f880c8719801f	c5e056176b8aca	fad11e12df9039
2aaa8e2b763284	5c984850af5937	9025f47efb9ee7	c7516de248ebaa	fbdcc02bed9524
2d25a5e0825a0a	5d9776266777d1	903803b6025871	c77d5e39c9fe01	fc1c6588c7fee1
2dde587cf0e579	5ff16d38bdc8fb	907037037094e6	c92384b5a33d51	fc3a839ddce9a6
2ea922659549b7	615983d3fa3dc0	9251cab8ab5ba3	caaf532c9028ac	
2fa253646985ae	619f526eeb6b5d	93eea84cd8aaf2	cdf7527ced93c9	

CHAPTER 8: Conclusion

8.1 Summary of Attacks

Table 8.1 summarizes the attacks on SoDark-3 presented in this thesis. For five rounds and fewer, key recovery attacks are possible, given two arbitrary plaintext–ciphertext–tweak tuples. The attacks have time complexities that are significantly lower than for exhaustive key search. Additionally, key recovery using SAT solvers is possible for four and fewer rounds.

Table 8.1. Summary of attacks on SoDark-3. Time complexities are weighted to be proportional to the brute force complexity of 2^{55} for the same number of rounds (see Section 4.1).

Section	Type	Rounds	Time	Data	Memory
4.3	Known-plaintext structural	2	2^9	2	$2^{4.1}$
4.4	Known-plaintext structural	3	2^{16}	2	–
4.5	Known-plaintext structural	4	$2^{32.9}$	2	$2^{17.6}$
4.6	Known-plaintext structural	5	2^{49}	2	2^8
4.7	Chosen-tweak structural	6	$2^{46.1}$	$2^{12.7}$	–
4.7	Chosen-tweak structural	7	$2^{46.1}$	$2^{12.7}$	–
4.8	Chosen-tweak structural	8	$2^{45.1}$	$2^{12.7}$	–
6	Known-plaintext SAT-based	≤ 4	Low	3	Low
7.4	Known-plaintext brute force	$\star$	2^{55}	2	–
7.5	Ciphertext-only brute force	$\star$	2^{55}	> 2	–

Attacks on six, seven, and eight rounds also exist with low time complexities. Their data complexities are manageable, but the requirements on relationships between tweaks make the attack hard to implement by a passive attacker. Referring back to Section 4.7, the attack requires all bytes in the tweaks in a pair of plaintext–ciphertext–tweak tuples to be identical, except for the fifth tweak byte. Considering the description of the ALE protocol in Section 2.3, this may indeed be possible to arrange for an attacker that, for example, has come in possession of a keyed ALE radio.

It should be noted that, if tweaks are generated in accordance with the specifications in [4], the fifth byte contains the word number, see Table 2.2. It will be the only byte that changes between PDUs in a single linking transmission. There is therefore a small chance that the output differentials required for the six-, seven-, and eight-round attacks will occur during normal operation. For ALE networks that use AL-1, this probability will be higher than for networks using AL-2, due to the longer PI.

In a normal three-PDU linking transmission, the PDUs form three different plaintext–ciphertext–tweak tuple-pairs, all with the required input differential. From Equation 4.80, the required number of intercepted linking transmissions required to find the correct output differential with 50% probability is therefore

$$\frac{11{,}629{,}080}{3} = 3{,}876{,}360. \tag{8.1}$$

To put the number in perspective, it is equivalent to intercepting a linking transmission every eight seconds for a year. This is obviously not a realistic setting, except for possibly in some very high intensity military operations. It should be considered, however, that given the high proliferation of ALE technology and considering all messages by all users worldwide, there is certainly a non-negligible probability of the output differential appearing somewhere within some sufficiently large time interval.

The demonstrated feasibility of brute force attacks on the SoDark ciphers, regardless of the number of rounds, shows that the level of protection provided by ALE linking protection is not sufficient. This is in agreement with the key length recommendations presented in Chapter 2.

8.2 Discussion

"Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break. It's not even hard. What is hard is creating an algorithm that no one else can break, even after years of analysis."

— Bruce Schneier [50]

A fundamental maxim in cryptography is that one should not use proprietary or "home-made" cipher algorithms in any setting that requires real security. The pitfalls in cipher construction are many and even world-leading experts have failed in such efforts. The accepted best practice is to use well-known algorithms that have been developed and vetted thoroughly [51]. AES is probably the best known example of a cipher that satisfies these requirements. For that reason, it should come as no surprise that it is the world's most used cipher algorithm.

With this in mind, the decision by the creators of the ALE standards to design their own cipher algorithm is unfortunate. At the time 2G ALE was standardized, DES—though also a 56-bit cipher—was well known and used. Together with a suitable block cipher mode of operation, it would have been a good candidate in lieu of Lattice/SoDark. In any case, with developments during the 1990s in both cryptanalysis and demonstrated exhaustive key searches performed by, among others, the EFF and Distributed.net, a replacement of the 56-bit linking protection algorithm should have been considered at the time.

The use of a tweak in SoDark to thwart replay attacks, which was novel for the time, should be noted. Not only does it fulfill the requirements of channel- and time-variation well, it also effectively prevents the construction of TMTO attacks to which other ciphers with weak structure and short key lengths are susceptible. While many design decisions made in the construction of the ALE linking protection ciphers can be criticized, the design and inclusion of a tweak is certainly not one of those.

The weaknesses presented in the SoDark cipher family and their impact on the ALE system as a whole is a good example of how design flaws in subsystems affect the design goals of the larger system. In this case, the design goals regarding confidentiality, integrity, and availability in the ALE system hinge completely on the cryptographic strength of the SoDark algorithm.

An attacker with knowledge of an ALE linking protection key can attack an ALE HF radio system in a number of ways: First, the attacker can compromise confidentiality by recovering encrypted plaintexts. This will include identities of senders and receivers as well as any orderwire traffic transmitted using the ALE protocol.

Second, the adversary can compromise the integrity of the network by injecting arbitrary ALE PDUs. This can be leveraged to establish links and inject higher level protocol traffic. The ability to inject PDUs can also be used to geographically locate other stations, by causing them to automatically transmit responses to received linking requests.

Third, availability attacks are possible through PDU injection. For example, by saturating an ALE network with link establishment calls, an adversary can tie up all radio stations in the network with fake traffic, preventing the transmission of real traffic.

The synchronous nature of 3G ALE makes it vulnerable to more attacks by an adversary with knowledge of the linking protection key. For example, by transmitting faked replies to time synchronization requests, the adversary can force radios out of the network by providing deliberately inaccurate time synchronization responses.

It is also worth emphasizing that ALE linking protection, whether the cipher is secure or not, only protects the linking process itself. After the link has been established, it is handed off for use by higher level protocols. If those protocols do not include protection mechanisms of their own, attacks on established links are possible without knowledge of the linking protection key through the use of normal electronic warfare traffic injection methods.

8.3 Recommendations

The ciphers in the SoDark family should not be used.

For short-term mitigation, ALE linking protection users should change keys at least on a daily basis, regardless of their threat model. If the threat model includes adversaries that have access to the resources of medium or large organizations, keys should be assumed to be recovered within, at most, hours from interception of traffic. Appropriate changes in operating procedures should be made to ensure protection of confidentiality, integrity, and availability in the system.

For long-term mitigation, the solution is to implement secure replacements for the SoDark ciphers. Users that have access to AL-3 and AL-4 linking protection ciphers can use those. For users that do not, a suggested replacement for SoDark is outlined in the next section.

8.4 A Suggested Replacement for SoDark

According to [52], the ALE designers are aware of the questionable security of the SoDark family. For that reason, they are considering a replacement cipher for fourth-generation (4G) ALE. Unfortunately, a purpose-made cipher, Halfloop, is once again a candidate, both to replace 24- and 48-bit SoDark in 4G ALE as well as in a 96-bit version for encryption of the 96-bit PDUs introduced in that standard.

A better option would be to use encryption based on best practice methods to replace SoDark in 2G and 3G ALE and for linking protection in 4G ALE.

AES is by far the most used and most trusted cipher algorithm today. It was created and standardized through an open and rigorous process. Additionally, it is the first, and so far only, publicly developed cipher approved by the NSA for protection of U.S. classified information.

With a block size of 128 bits, AES cannot be applied as a drop in replacement. However, using block ciphers directly is unusual in applications. This is the purpose of block cipher modes of operation. A mode of operation that preserves the format of the encrypted PDUs as well as satisfies the other requirements on linking protection is the Thorp shuffle [53]. It stands on a sound mathematical foundation and is backed by solid reasoning concerning its security. It is well suited for format preserving encryption of the small blocks used in the ALE standards.

The Thorp shuffle is a maximally unbalanced Feistel network that encrypts a single bit per round, so the number of rounds is equal to the block size. Figure 8.1 illustrates one round of the Thorp shuffle. Here, AES is suggested as a round function. The authors of [53] present a method to avoid calling the round function in every round that they dub the *5x trick*. Using this method, the function only needs to be called $\lceil \frac{24}{5} \rceil = 5$ times in the 24-bit case, $\lceil \frac{48}{5} \rceil = 10$ times in the 48-bit case, and $\lceil \frac{96}{5} \rceil = 20$ times in the 96-bit case. The number of passes of the Thorp shuffle required for proper security is investigated in [53].

Since $n - 1$ bits of input to the AES round function are used, where n is the block size, the remaining $129 - n$ bits can be used to input a tweak. In the case of $n = 96$, only 33 bits are available for tweak use. A solution to this could be to use an additional AES encryption

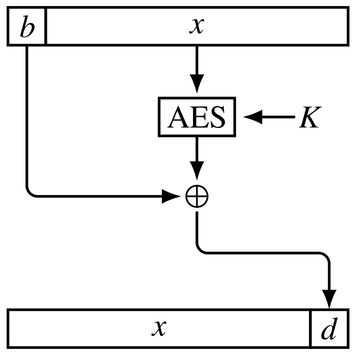

Figure 8.1. One round of the Thorp shuffle with AES as the Feistel round function. One bit, b is encrypted into a bit d and concatenated with the unaffected bits x. Adapted from [53].

operation to compress the 64-bit tweak and add the result to the input in some suitable manner.

One of the reasons the Rijndael algorithm was selected for the AES standard over the other candidates was its speed on a variety of platforms, including on small 8-bit embedded systems [16]. This, together with the low number of PDUs encrypted in any linking operation, should make the speed of the proposed solution acceptable, even on embedded hardware in field radios.

8.5 Ideas for Further Research

Many lines of effort were abandoned due to time constraints. They may provide further insight into the security of the SoDark family of ciphers.

Structural attacks, like the ones described in Chapter 4 may be possible for more than eight rounds. The filtering technique described in Sections 4.7 and 4.8 that enables identifying specific differentials many rounds into the cipher with high probability works on any number of rounds.

No structural attacks were attempted on SoDark-6. The methods developed for SoDark-3 are likely applicable and may yield similar results.

Approaches to algebraic cryptanalysis other than the one used in Chapter 6 may prove fruitful. For example, SAT solvers based on belief propagation tend to be very fast in solving known satisfiable SAT problems. In some cases they are able to solve very large problems where other SAT solvers fail [44].

The algorithms used to create the logic circuit representations were designed for creating circuits that are efficient to implement on modern CPUs. Modification of the algorithms so that they can find networks with all 14 non-trivial Boolean functions of two variables would likely result in smaller circuits that are easier for SAT solvers to handle.

An extension of the brute force solver developed in Chapter 7 to handle the ciphertext-only attacks described in Section 7.5 would provide an upper bound on the security of the cipher in best-case conditions.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] E. E. Johnson, E. Koski, W. N. Furman, M. Jorgenson, and J. Nieto, *Third-generation and Wideband HF Radio Communications*. Norwood, MA: Artech House, 2012.

[2] List of automatic link establishment agencies and frequencies. (2017). [Online]. Available: http://www.ominous-valve.com/ale-list.txt.

[3] B. Schneier, "A self-study course in block-cipher cryptanalysis," *Cryptologia*, vol. 24, no. 1, pp. 18–33, 2000, doi:10.1080/0161-110091888754.

[4] *Interoperability and Performance Standards for Medium and High Frequency Radio Systems*, United States Department of Defense Std. MIL-STD-188-141C, 2011.

[5] E. E. Johnson, "A 24-bit encryption algorithm for linking protection," USAISEC, Tech. Rep. ASQB-OSO-S-TR-92-04, Mar. 1992.

[6] D. Gollman, *Computer Security*, 2nd ed. West Sussex, UK: John Wiley & Sons Ltd., 2006.

[7] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. New York, NY: CRC Press, 1996.

[8] A. Kerckhoffs, "La cryptographie militaire [military cryptography]," *Journal des sciences militaires [Journal of Military Sciences]*, pp. 5–83, 1883.

[9] C. E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949, doi:10.1002/j.1538-7305.1949.tb00928.x.

[10] R. Schroeppel and H. Orman. (1998, July). An Overview of the Hasty Pudding Cipher. [Online]. Available: http://richard.schroeppel.name:8015/hpc/hpc-overview.

[11] M. Liskov, R. L. Rivest, and D. Wagner, "Tweakable block ciphers," in *22nd Annual International Cryptology Conference*, Santa Barbara, CA, 2002, pp. 31–46, doi:10.1007/3-540-45708-9_3.

[12] Institute for Telecommunication Sciences. Voice of America Coverage Analysis Program (VOACAP). [Online]. Available: https://www.its.bldrdoc.gov/resources/radio-propagation-software/high-frequency/high-frequency-propagation-models.aspx.

[13] *Telecommunications: HF Radio Automatic Link Establishment*, National Communications System – Office of Technology and Standards Std. FS-1045A, 1993.

[14] A. Bogdanov, D. Khovratovich, and C. Rechberger, “Biclique cryptanalysis of the full AES,” in *17th International Conference on the Theory and Application of Cryptology and Information Security*, Seoul, South Korea, 2011, pp. 344–371, doi:10.1007/978-3-642-25385-0_19.

[15] G. Bard, *Algebraic Cryptanalysis*. Dordrecht, Netherlands: Springer Science & Business Media, 2009, doi:10.1007/978-0-387-88757-9.

[16] L. R. Knudsen and M. Robshaw, *The Block Cipher Companion*. Heidelberg, Germany: Springer Science & Business Media, 2011, doi:10.1007/978-3-642-17342-4.

[17] N. Smart *et al.*, “ECRYPT II yearly report on algorithms and keysizes (2011–2012),” Katholieke Universiteit Leuven, Tech. Rep. D.SPA.20, Sep. 2012.

[18] The Electronic Frontier Foundation, *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*, 1st ed. Sebastopol, CA: O’Reilly & Associates Inc., 1998.

[19] E. Biham, “A fast new DES implementation in software,” in *4th International Workshop on Fast Software Encryption*, Haifa, Israel, 1997, pp. 260–272, doi:10.1007/BFb0052352.

[20] M. Hellman, “A cryptanalytic time-memory trade-off,” *IEEE Transactions on Information Theory*, vol. 26, no. 4, pp. 401–406, July 1980, doi:10.1109/TIT.1980.1056220.

[21] A. Biryukov, A. Shamir, and D. Wagner, “Real time cryptanalysis of A5/1 on a PC,” in *7th International Workshop on Fast Software Encryption*, New York, NY, 2000, pp. 1–18, doi:10.1007/3-540-44706-7_1.

[22] W. Diffie and M. E. Hellman, “Special feature exhaustive cryptanalysis of the NBS data encryption standard,” *Computer*, vol. 10, no. 6, pp. 74–84, 1977, doi:10.1109/C-M.1977.217750.

[23] E. Biham and A. Shamir, “Differential cryptanalysis of DES-like cryptosystems,” in *Conference on the Theory and Application of Cryptography*, Santa Barbara, CA, 1990, pp. 2–21, doi:10.1007/3-540-38424-3_1.

[24] M. Matsui, “Linear cryptanalysis method for DES cipher,” in *Workshop on the Theory and Application of Cryptographic Techniques*, Lofthus, Norway, 1993, pp. 386–397, doi:10.1007/3-540-48285-7_33.

[25] H. M. Heys, “A tutorial on linear and differential cryptanalysis,” *Cryptologia*, vol. 26, no. 3, pp. 189–221, 2002, doi:10.1080/0161-110291890885.

[26] A. Biryukov and L. Perrin, "On reverse-engineering S-boxes with hidden design criteria or structure," in *Conference on the Theory and Application of Cryptography*, Santa Barbara, CA, 2015, pp. 116–140, doi:10.1007/978-3-662-47989-6_6.

[27] L. Perrin and A. Udovenko, "Algebraic insights into the secret Feistel network," in *23rd International Conference on Fast Software Encryption*, Bochum, Germany, 2016, pp. 378–398, doi:10.1007/978-3-662-52993-5_19.

[28] M. Soos, "The CryptoMiniSat 5 set of solvers at SAT Competition 2016," in *SAT Competition 2016*, Bordeaux, France, 2016, p. 28, doi:10138/164630.

[29] A. Biere, "Splatz, Lingeling, Plingeling, Treengeling, YalSAT entering the SAT Competition 2016," in *SAT Competition 2016*, Bordeaux, France, 2016, pp. 44–45, doi:10138/164630.

[30] S. Even and Y. Mansour, "A construction of a cipher from a single pseudorandom permutation," in *International Conference on the Theory and Application of Cryptology*, Fujiyoshida, Japan, 1991, pp. 210–224, doi:10.1007/3-540-57332-1_17.

[31] J. Daemen, "Limitations of the Even–Mansour construction," in *International Conference on the Theory and Application of Cryptology*, Fujiyoshida, Japan, 1991, pp. 495–498, doi:10.1007/3-540-57332-1_46.

[32] O. Dunkelman, N. Keller, and A. Shamir, "Minimalism in cryptography: The Even-Mansour scheme revisited." in *31st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Cambridge, UK, 2012, pp. 336–354, doi:10.1007/978-3-642-29011-4_21.

[33] I. Dinur, O. Dunkelman, N. Keller, and A. Shamir, "Key recovery attacks on 3-round Even–Mansour, 8-step LED-128, and full AES^2," in *19th International Conference on the Theory and Application of Cryptology and Information Security*, Bengaluru, India, 2013, pp. 337–356, doi:10.1007/978-3-642-42033-7_18.

[34] I. Dinur, O. Dunkelman, N. Keller, and A. Shamir, "Cryptanalysis of iterated Even–Mansour schemes with two keys," in *20th International Conference on the Theory and Application of Cryptology and Information Security*, Kaoshiung, Taiwan, R.O.C., 2014, pp. 439–457, doi:10.1007/978-3-662-45611-8_23.

[35] I. Dinur, O. Dunkelman, N. Keller, and A. Shamir, "Key recovery attacks on iterated Even–Mansour encryption schemes," *Journal of Cryptology*, vol. 29, no. 4, pp. 697–728, 2016, doi:10.1007/s00145-015-9207-3.

[36] A. Bogdanov, L. R. Knudsen, G. Leander, F.-X. Standaert, J. P. Steinberger, and E. Tischhauser, "Key-alternating ciphers in a provable setting: Encryption using a small number of public permutations," in *31st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Cambridge, UK, 2012, pp. 45–62, doi:10.1007/978-3-642-29011-4_5.

[37] A. Bogdanov and C. Rechberger, "A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN," in *17th International Workshop on Selected Areas in Cryptography*, Waterloo, Ontario, Canada, 2010, pp. 229–240, doi:10.1007/978-3-642-19574-7_16.

[38] M. Dansarie. (2017, Sep.). SoCracked. [Online]. Available: https://github.com/dansarie/SoCracked. doi:10.5281/zenodo.893134.

[39] M. Kwan. (2000, Oct.). Reducing the gate count of bitslice DES. Cryptology ePrint Archive, Report 2000/051. [Online]. Available: https://eprint.iacr.org/2000/051.

[40] DeepLearningJohnDoe. (2015). SBOXDiscovery. [Online]. Available: https://github.com/DeepLearningJohnDoe/SBOXDiscovery.

[41] K. H. Rosen, *Discrete Mathematics and Its Applications*, 7th ed. New York, NY: McGraw-Hill, 2012.

[42] M. Dansarie. (2017, Sep.). sboxgates. [Online]. Available: https://github.com/dansarie/sboxgates. doi:10.5281/zenodo.891021.

[43] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphviz—open source graph drawing tools," in *9th International Symposium on Graph Drawing*, Vienna, Austria, 2001, pp. 483–484, doi:10.1007/3-540-45848-4_57.

[44] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Boston, MA: Pearson Education Inc., 2015.

[45] Wikipedia. (2017). Tseytin transformation. [Online]. Available: https://en.wikipedia.org/wiki/Tseytin_transformation.

[46] SAT Competitions. The international SAT Competitions web page. [Online]. Available: http://www.satcompetition.org/.

[47] N. Eén and N. Sörensson, "An extensible SAT-solver," in *6th International Conference on Theory and Applications of Satisfiability Testing*, Santa Margherita Ligure, Italy, 2003, pp. 502–518, doi:10.1007/978-3-540-24605-3_37.

[48] Nvidia Corp. (2017, June). Parallel Thread Execution ISA. [Online]. Available: https://docs.nvidia.com/cuda/pdf/ptx_isa_5.0.pdf.

[49] Nvidia Corp. (2017, June). CUDA C programming guide. [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[50] B. Schneier, “A memo to the amateur cipher designer,” *Crypto-Gram*, no. 10, 1998, https://www.schneier.com/crypto-gram/archives/1998/1015.html#cipherdesign.

[51] B. Schneier, *Applied Cryptography*, 2nd ed. New York, NY: John Wiley & Sons Ltd., 1996.

[52] E. E. Johnson, “Wideband ALE – the next generation of HF,” in *2016 Nordic HF Conference, HF 16*, Fårö, Sweden, 2016.

[53] B. Morris, P. Rogaway, and T. Stegers, “How to encipher messages on a small domain,” in *29th Annual International Cryptology Conference*, Santa Barbara, CA, Aug. 2009, pp. 286–302, doi:10.1007/978-3-642-03356-8_17.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California